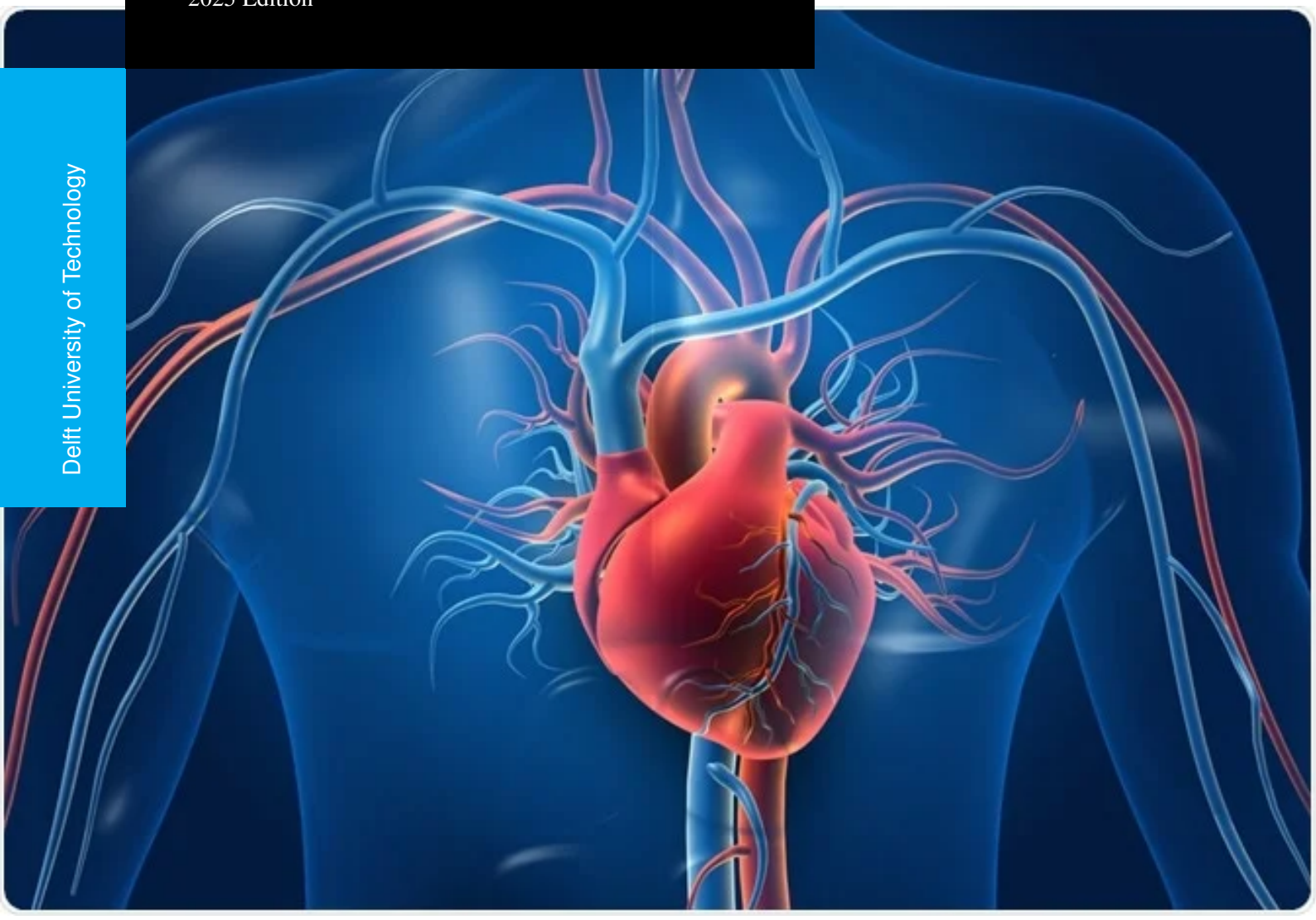
EE2L1

Manual

IP3: Heart Sound Localization

2025 Edition



Delft University of Technology

Prof.dr.ir. Alle-Jan van der Veen
Dr. Bahareh Abdikivanani

TU Delft Delft University of Technology

Challenge the future

The IP3 project “Heart sound localization” was conceived in 2024 as a second-year BSc-EE lab project. The topic was inspired by the work of Hanie Moghaddasi. The following people have contributed to it:

Bahareh Abdikivanani
Hendrik Bosma
Gerard Janssen
Nick van der Meijs
Alle-Jan van der Veen

Koen Bavelaar
Nassim Beladel
Elena van Breukelen
Finn Heijink
Ojasvi Kumar

Contents

Chapter 1

INTRODUCTION

Contents

The iconic image of a doctor is a person in a white coat, with a stethoscope around the neck. Besides as status symbol, the stethoscope has served very well over the past 200 years to quickly assess vital functions of patients. But today, in the 21st century, it is necessary to go digital. Fortunately, this is rather straightforward: just insert a microphone, or even a mobile phone can be used to do the recording.

What is better than a single microphone? Think about two, four, or why not six? The signals of the microphones can be added to average out noise, but more is possible.

Welcome to the EE2L1 IP3: Heart Sound Localization Challenge. In this project, you and your team are going to use a 6-element digital stethoscope to do signal analysis and in particular: to attempt to localize the heart valves responsible for the sounds. Signals from different directions arrive at slightly different times at the microphones, and from this their directions can be estimated.The same principles are used in many imaging systems, such as ultrasound echoscopy, which is also sound-based but at 1–10 MHz, rather than 0–1 kHz. The general term is “interferometry” or “array processing”, and it is used in radio astronomy, radar, medical imaging, wireless communication (MIMO), geoscience, etc.

As in previous projects, the objective of IP3 is to apply and integrate the EE knowledge you have acquired so far. For this project, this mostly entails signal transforms and linear algebra. The project also prepares you for future courses such as Signal Processing, Systems and Control, Telecommunication, and electives such as Machine Learning.

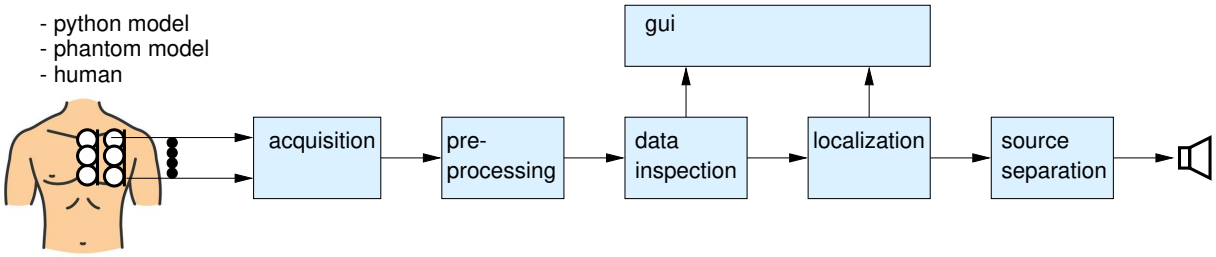


Figure 1.1. System overview

1.1 OVERVIEW

1.1.1 Scope

The primary objective of the project is to record and analyze phono-cardiogram (PCG) signals using a microphone array for the purpose of localizing and separating the internal sound sources within the heart. By capturing these acoustic signals, our aim is to enhance diagnostic capabilities, particularly in the identification of heart valve disorders. After all, an electronic stethoscope avoids subjective factors related to the ear sensitivity and experience of individual physicians, and enables to automate the interpretation of the acoustic properties of the heart.

In this project, you work with a 6-channel digital stethoscope array. These microphones are strategically positioned on the human chest to optimize the capture of heart sounds. Following the acquisition of PCG signals, the data undergoes preprocessing and segmentation to ensure accuracy and reliability in subsequent analysis stages. The next objective is to localize the sources (e.g., the 4 valves) using deconvolution, direction finding, and MUSIC-like (eigenvalue-based) algorithms. The algorithms are designed and tested on software models and a phantom model (dummy). After consideration of the ethical aspects, the signals could also be acquired on human subjects.

This is a complex assignment, which is therefore split into several modules:

- *Module 1: Single channel heart sound analysis.* Study biological aspects of the heart (systole and diastole phases), listen to heart sounds, and apply basic preprocessing and analysis tools, such as lowpass filtering and spectrograms. Experiment with single-channel recordings.
- *Module 2: Multichannel heart sound modeling.* Clean multichannel data is not readily available! Create a simulated data model to generate such data, with known parameters.
- *Module 3: Basic direction finding (1D).* Study the basics of array processing, where the goal is to find the direction of a single source in a 1D setting. Develop algorithms such as Matched Filter and MVDR. Experiment with a 6-element linear array.
- *Module 4: Advanced direction finding (1D).* Apply the SVD to find the number of sources. Develop an advanced direction finding algorithm (MUSIC). Experiment with 2 sources and compare MF, MVDR and MUSIC.

- *3D localization.* Extend the MUSIC algorithm to near-field processing (3D). Use simulate data from module 2 to test it.
- *Integration and Testing.* Combine the signal analysis algorithms with the MUSIC algorithm. Test on the simulated data model. Experiment with the stethoscope array on a dummy and on persons (or use given real test data).

The completed system could look like Figure 1.1. The final block "source separation" will not be included; it would allow to listen to each heart valve separately.

1.1.2 Hardware

The project will use the following hardware:

- Stethoscope chest pieces with integrated microphones (home-made), see Fig. 1.2(*a*);
- Stethoscope chest pieces, for assembly in a 2×3 array;
- 8-channel, 96 kHz amplifier/data recorder Presonus AudioBox 1818VSL, capable to drive loudspeakers and simultaneously record microphones, see Fig. 1.2(*b*);
- Raspberry Pi 5 to interact with the sound card and make recordings as well as playing sound at the same time; see Fig. 1.2
- Test setup with a 6-microphone linear array, for direction finding experiments, see Fig. 1.2(*c*);
- Test setup (phantom model) consisting of a box with 2 loudspeakers and a thick layer of foam, mimicking the human body, see Fig. 1.2(*d*).

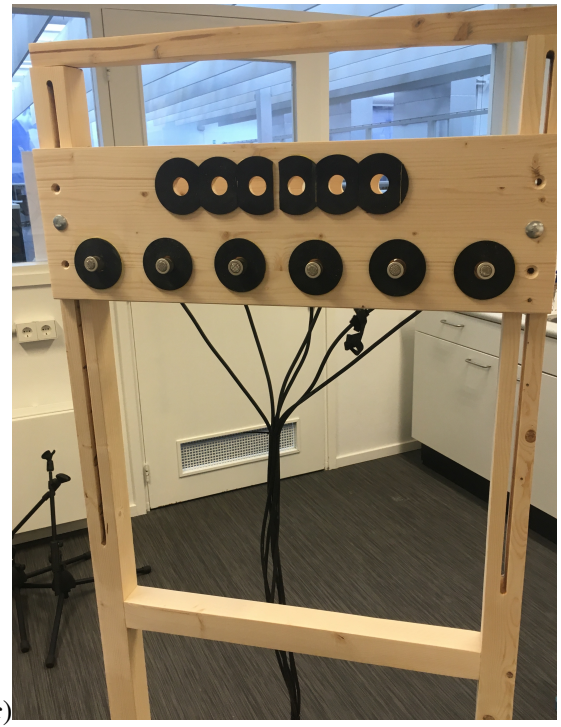
1.2 EDUCATIONAL OBJECTIVES

General learning objective: To integrate different technical areas related to electronic systems, signal processing and control. The educational objectives are:

- Analyze a real-world application which involves both hardware and software by breaking it down into smaller modules.
- Apply signal processing (and linear algebra, statistics, modeling, communication, control) tools to design and implement the solution using the python programming language.
- Increased skills in building and testing electronic systems.
- Increased skills in measurement techniques, e.g., acoustic channel measurements.

(*a*)

(*b*)



(*c*)

(*d*)

(*e*)

Figure 1.2. Available hardware: (a) digital stethoscopes, (b) multichannel recorder, (c) linear microphone array, (d) phantom heart model, (e) Raspberry Pi 5.

- Design a complex system using design methodology as developed over the various IP labs.
- Plan, manage and conduct effective teamwork for the successful realization of a technical complex assignment.
- Write and present a technical report that complies with current standards of the electrical engineering scientific community.
- Reflect on ethical aspects associated with the application of the designed system in society.

The main difficulties students have with this project are the complexity of the overall system, and planning. You will split up your team into two sub-groups. Each delivers Python code that later has to be integrated and function together. Nobody in the team has a complete detailed knowledge of the entire design. If you manage this then you will pass the project successfully. Essential parts to make this work are:

- *System engineering:* At an early stage, agree on an overall structure for your design, partition into modules, and decide on specifications for the modules. Find the essential aspects that need to be done right.
- *Testing:* Each module should have a clearly defined specification, and the Python code should be tested and verified against the specifications before it is integrated into the overall system. You cannot debug the overall system if you are unsure about its parts!

To help you with teamwork, in Week 2 there will be an ITAV session that will familiarize you with Scrum, i.e., teamwork methods used in the ICT industry for rapid design cycles.

1.3 SCHEDULE AND DEADLINES

While the exact schedule is on Brightspace, table 1.1 shows a generic schedule. The planning regarding the completion of the modules is flexible, but the deadlines are firm.

We will have a limited number of set-ups of the microphone systems. There are many groups competing for time on these systems. You will need to reserve time slots on one of these systems using a planner given in Brightspace.

1.4 ASSESSMENT AND REPORTS

1.4.1 Mid-term assessment and report

In weeks 4 and 5, you are asked to demonstrate the functionality and performance of your signal analysis and direction finding scripts to your TA. After their approval, you are ready to write your mid-term report, detailing your approach, implementation, and test results.

Table 1.1. Generic schedule and deadlines

Date	**Description**	
Week 1	Preparatory Colab assignment sessions	
Week 2	Kick-off and Introduction	
Week 3,4	M1: Single channel processing	M3: Basic direction finding
	M2: Heart sound modeling	M4: SVD and the MUSIC algorithm
Week 5	**Midterm report deadline**	
Week 6,7	3D localization, Phantom setup, algorithm extension, and system integration	
Week 8	Final report writing	
Week 9	**Final demo + Final report deadline**	
Week 10	**Presentation and discussion**	

Mid-term report You submit a report of approximately 15 pages (plus cover page and appendix) documenting your work until now. More details on the structure and contents of this report are in Chapter 8.

1.4.2 Final demo

In weeks 8 and 9, you will receive further instructions about the demo, during which the performance of your algorithms will be assessed.

1.4.3 Final report and presentation

Final report The project outcome is documented in a final report. The report must follow a structured approach, which you have learned in previous projects; the midterm report is a part of it. Use a to-the-point, concise but complete reporting style. Provide an appendix with all Python code. More details on this report are in chapter 11.

In the final report, you also briefly discuss ethical issues (from an engineering perspective).

The report without appendix should be about 30 pages. Within these pages you must document your design choices, explain your control systems, and report the deliverables of all modules, how you combined these modules and what problems you ran into and how you solved them. The focus is on your findings and measurement results and the corresponding conclusions. You are also judged regarding project skills such as planning and teamwork.

Final presentation and discussion In week 10 (consult Brightspace for the exact date), you present and defend your final report before an examination committee. The examiners will ask questions about your design choices and aspects of teamwork. This will be part of your grade.

The presentation lasts at most 10 min. Focus on the highlights and special features of the design, and mention the work breakdown and distribution of tasks to team members.

While each team member may not have been directly involved in every aspect of the project, through open communication and collaboration, all team members are expected to have an operating understanding of all parts of the project. During the discussion, detailed questions will be asked to the group. The team member with the best subject knowledge is encouraged to answer. However, less complex questions can be asked to anyone to test their participation in the project.

The examination will last about 30 min. After the examination, you will be asked to fill in a peer review form. Individual grades are differentiated depending on staff observations and the outcome of the peer review.

1.4.4 Grading

Your grade depends on the following:

- Midterm report (30%);
- Performance during the demos and tests (20%);
- Final report, oral presentation and defense (50%).

Teamwork is important. Your individual grade may differ depending on staff observations and a peer review system. There is a penalty for submitting the report late.

If your final grade is insufficient, you may have a chance to improve your grade by improving your reports.

1.5 FACILITIES

The project is carried out at the Tellegen Hall facilities in A and B teams. Each team has 2 sessions per week. The test setups are shared by 6 teams.

The following support is available:

- *Student assistants*; student assistants are your primary help. Each assistant supports up to four teams. Assistants also check attendance and progress.
- *Instructors*; Practicum coordinators are available at each lab session. The coordinators also grade your reports.
- *Technical support*; for questions about hardware and implementation issues, you can contact the student assistants and/or the technicians at the facilities.

Visit the Brightspace page of the course for support and contact information.

1.6 RULES AND REGULATIONS

In addition to the rules and regulations of the EEE group on the use of the Tellegen Hall, the following rules and regulations are applicable:

- You are expected to be present during your scheduled lab sessions. If you cannot attend for a good reason, contact us *before* the lab session. Absence more than two times will not be allowed; you will be removed from the practicum.
- Preparation for labdays is *mandatory*. This project has an intense pace and limited time; use it wisely. The scheduled homework time is needed.
- You may not work alone in the lab. A student assistant or staff member must be present.

You are working with stethoscopes and may acquire personal data. This should always be voluntary. You are expected to treat the data with due respect for privacy, e.g., anonymize the files.

1.7 INSTALLING PYTHON

Before we begin, you'll need to set up your programming environment in Python. This includes installing an Integrated Development Environment (IDE) and the necessary packages for the project. While you are free to use any software (including Google Colab), due to the scope of the project it is recommended to work locally and set up a GitHub or GitLab repository for version control and collaboration.

1.8 GETTING TO KNOW YOUR TEAM

To form a team with new people is not always easy. Let's start with a simple task: filling out a document together. Please look on Brightspace for the document `ip3_kickoff.docx`. The document asks questions about how you want to collaborate as a team. Submit the document in your Brightspace assignment folder.

As a team, you should find ways to brainstorm as a team, find possible approaches to a problem (analyze the problem first!), and keep everyone busy, involved, and synchronized.

One thing to decide is to form two sub-groups: one that will work on signal analysis (Modules 1 and 2) and the other to work on localization (Modules 3 and 4). After these modules are finished, the mid-term report must be written according to the guidelines in Chapter 8. Please read Chapter 2 first and then complete and submit the kick-off form.

To help you improve your teamworking skills, in week 2 an ITAV session is scheduled which will familiarize you with Scrum. This is a framework for rapid design cycles as used in the ICT industry. We expect that this will be a useful skill for IP3 (and for your future career).

Chapter **2**

CARDIAC PHYSIOLOGY

Contents

An audio recording of the heart using a stethoscope is called a phono-cardiogram (PCG). It is a non-invasive measurement which is easily acquired, and complements a recording of the electrical signals (ECG). To understand the information that it gives, we have to explain the basics of cardiac physiology.

2.1 PUMP MECHANICS

In essence, the heart consists of two pumps, apposed to each other. A schematic view of one of the pumps is shown in Fig. 2.1. The pump is a hollow muscle and consists of two chambers: the atrium and the ventricle, and two valves, which allow the blood to flow in only one direction. The veins drain into the atrium, which fills up. At some point, the valve towards the ventricle opens, the atrium contracts, and the ventricle is filled. Next, the atrio-ventricular valve closes, the ventricular-artery valve opens, the ventricle contracts, and blood is ejected into the artery. The latter valve closes and the cycle repeats.

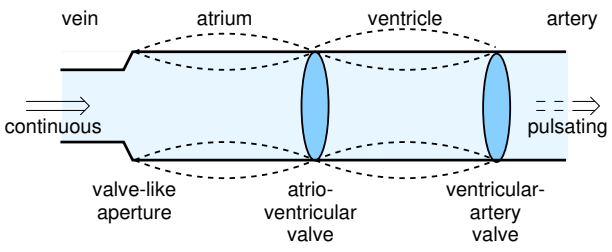


Figure 2.1. Two-chamber serial pump

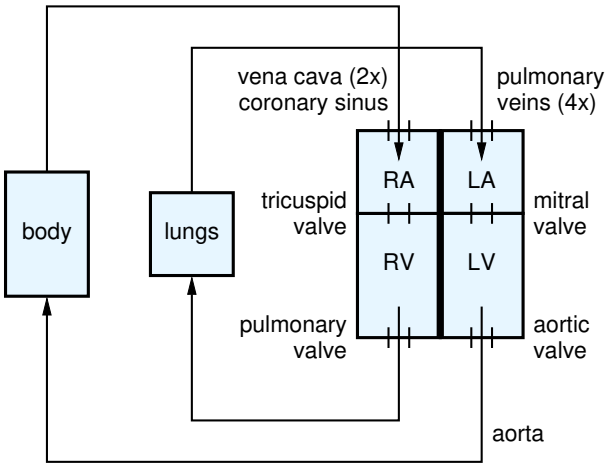


Figure 2.2. Schematic view of the circulation system

The heart contains two of these pumps, and this is shown schematically in Fig. 2.2. The veins connect to the right atrium (RA), and the right ventricle (RV) pumps blood to the lungs. After blood is enriched with oxygen, it arrives via the pulmonary veins into the left atrium (LA), and the left ventricle (LV) pumps blood into the aorta, which circulates it into the entire body. The four related valves are called tricuspid valve ("three leaves"), pulmonary valve, mitral valve (or bicuspid valve, "two leaves") and the aortic valve.

The two pumps act synchronously: both atria and both ventricles pump at the same time. The muscle contraction is controlled by electrical signals which propagate over the muscles, they can be measured using an electro-cardiogram (ECG). The contraction wave is initiated in a corner of the right atrium in the sino-atrial (SA) node, which is the natural pacemaker; this is visible in the ECG as the P-wave. The wave travels to the atrial-ventricular node (AV node) which delays the signals. After a while, the AV node distributes the wave over the ventricles, which strongly contract; this is visible as the QRS complex. At the end of the T-wave, the ventricles relax.

Fig. 2.3 shows the cardiac cycle in more detail. It has 4 phases:

- During diastole, the tricuspid and mitral valves are open and the ventricles fill up. In majority, this occurs by passive filling (from the pressure of blood in the veins), and this process can take up to 400 ms; less if your heart rate goes up. The atrium acts as a reservoir. In the final stage, the atrium contracts and delivers a final quantity of blood into the ventricle.
- The systolic phase starts by the closing of the tricuspid and mitral valves. In the PCG, this is audible; the sound is called the S1 signal, and it sounds as "lub". In the ECG, the R-peak is visible.
- Shortly after, the ventricles contract and the pulmonary valve and aortic valves open. Blood is ejected into the arteries.

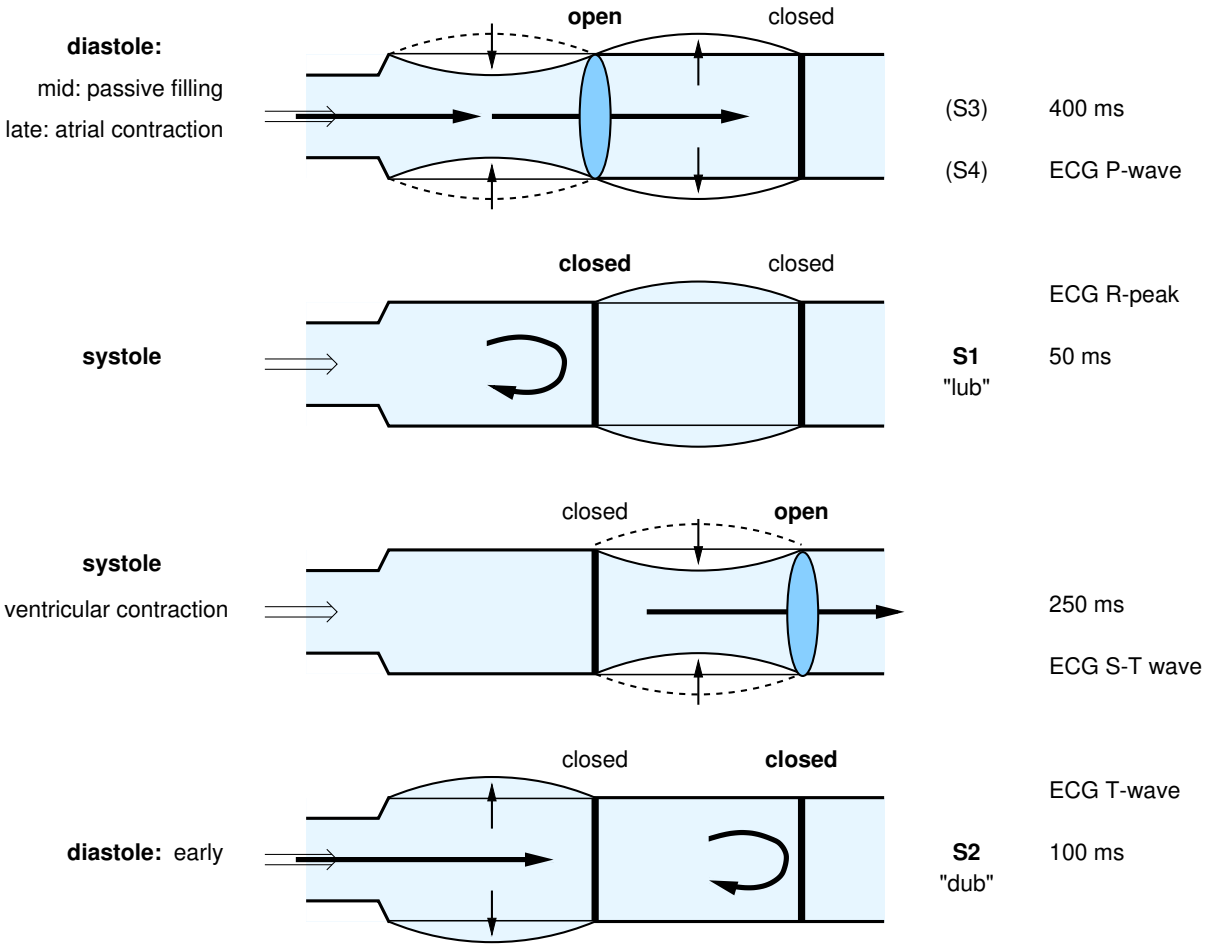


Figure 2.3. The cardiac cycle

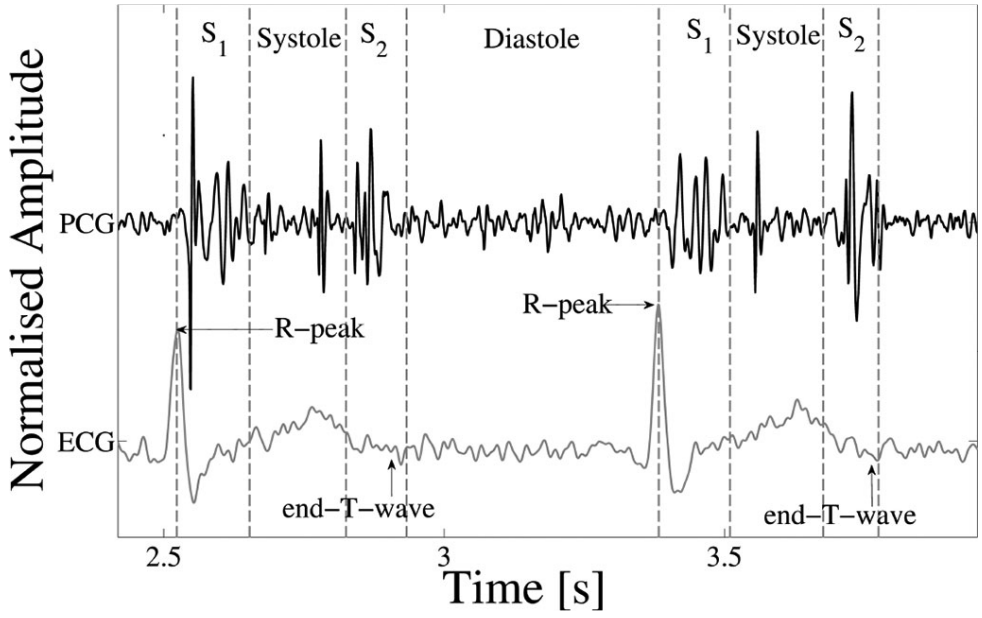


Figure 2.4. Example of a PCG along with the corresponding ECG. Also visible are midsystolic clicks, typical of mitral valve prolapse. Source: [1]

- Finally, the pulmonary and aortic valves close; this is audible as the S2 signal, which sounds like "dub". This is the early diastolic phase.

2.2 EXAMPLE OF A PCG

The acoustic signals coming from the heart are mostly originating from the valves and are in the range of 20–500 Hz.

An example PCG is shown in Fig. 2.4, along with the corresponding ECG. Under normal conditions, only the S1 and S2 sounds are audible. The S1 tone (10–200 Hz) is caused by the nearly simultaneous closing of the mitral and (shortly later) the tricuspid valves. It is stronger if the heart is working harder. The S2 sound could be split into two pulses that are closely following each other, the first due to the aortic valve and the second due to pulmonary valve. Nonetheless, the S2 is shorter in duration than S1, and also its pitch could be higher (20-250 Hz). Two other tones are of very low frequency and therefore usually inaudible: S3 (filling of the ventricles) and S4 (contraction of the atria).

The period between S1 and S2 (the systole) is usually shorter than the period between S2 and S1 (the diastole), and if your heart rate goes up, the diastolic period is reduced. The amplitudes are very much location-dependent. The figure also shows that, generally, a lot of background noise is present. This could come from the heart itself (e.g., blood flow), the lungs, or the abdomen. The microphones also

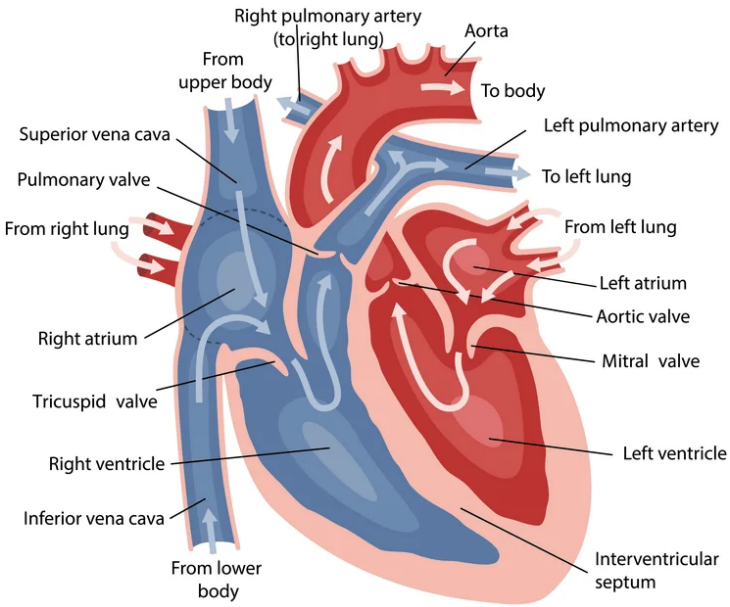


Figure 2.5. Anatomy of the heart

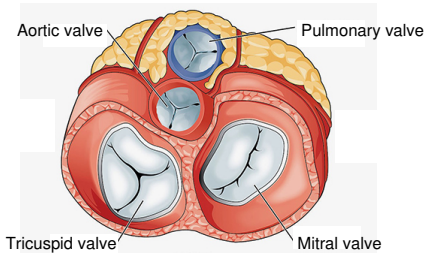


Figure 2.6. Vertical cross section ("coronal plane") showing valve locations

tend to pick up a lot of environment noise.

2.3 ANATOMY

The actual anatomy of the heart is shown in Fig. 2.5. The atria are above, and mostly behind the ventricles, which lie in a slightly tilted "V" shape. Their exits, i.e. the pulmonary and aortic valves, are in fact on the top. In a vertical cross section, it is seen that the pulmonary valve and aortic valve are located more or less in the same plane as (or slightly behind) the tricuspid valve and mitral valve (Fig. 2.6). They are also slightly higher.

The heart is located in between the lungs, behind the sternum (i.e. the breastbone that connects the ribs in the middle) and slightly to the left, as shown in Fig. 2.7. The figure also shows the areas where doctors listen to heart sounds (the auscultation locations).

The preferred aortic auscultation area is on the 2nd intercostal space (i.e. the gap between the ribs) at the right sternal border. The pulmonic area is on the 2nd intercostal space at the left sternal border. The tricuspid area is on the 5th intercostal space at the left sternal border, and the mitral area is on the 5th intercostal space midway the left part of the chest (where is the apex of the heart). Note that these locations do not directly correspond to the locations of the valves, but to the places where they are best

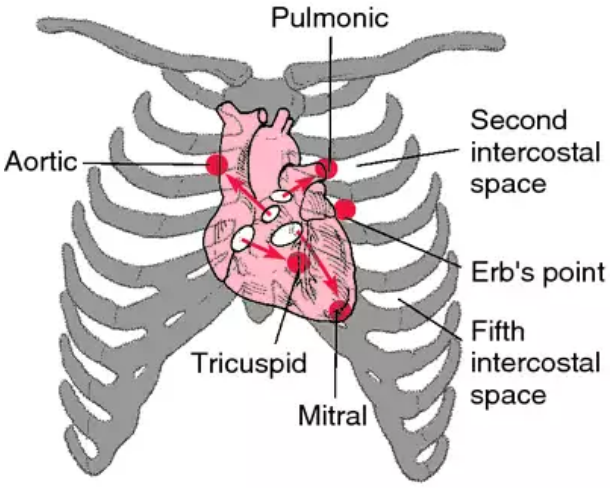


Figure 2.7. Auscultation locations, shown as red dots (white ovals indicate the corresponding valves)

heard.

The S1–S4 sounds are best heard around Erbs point, which is the 3rd intercostal space at the left sternal border. *For IP3, we recommend to measure around this point.*

Many videos can be found on youtube explaining where to listen and how to interprete what to hear; e.g. https://www.youtube.com/watch?v=wYZbMoWjLEg.

2.4 PATHOLOGY AUDIBLE IN THE PCG

Medical conditions that are audible in the PCG are usually connected to problems with one of the valves, in particular the valves on the left side (mitral and aortic valve) are of interest as this part does most of the work and pressures are higher.

- S3 and S4 are normally inaudible (20 Hz). Under pathological conditions, they become audible, and this is called ventricular and atrial gallop, respectively.
- The valves could be "leaky" and not completely close. In this case, a small amount of blood will continue to flow, which is called regurgitation. This is audible as a continuous murmur (60–600 Hz), after the S1 or S2 signal, possibly preceded by a "click".
- The valves could be constricted, this is called stenosis. The blood flow through the restricted area is faster, and after the valve it becomes turbulent, which is audible as a murmur (60–600 Hz). It is preceded by a "click".

- Other sounds could be due to a hole in the wall that separates the RV and the LV, or a connection between the aorta and the pulmonary artery.

In this project we do not aim to detect any anomalies. None of us is trained or qualified to make medical observations.

2.5 HOW TO USE A STETHOSCOPE

- Use a quiet room, minimizing ambient noise by closing doors and not talking.
- The subject can be examined in three positions: sitting, supine (i.e., lying face upwards), and left lateral recumbent. Some results will depend on position. For measuring heart signals, the supine position is preferred.
- The subject should breathe lightly (avoid lung sounds) or perhaps hold their breath after exhaling. Some results will depend on inhaling or exhaling.
- Stethoscopes should be used on the bare skin. Place the stethoscope chestpiece in direct contact with the chest wall. Use your index and middle finger to apply some pressure; avoid touching the tubing.
- Keep the stethoscope head still. Even the slightest motion gives rise to friction noise artefacts. Avoid chest hair.
- A standard stethoscope has two sides: a bell and diaphragm. The bell is the smaller side and is used for low-frequency sounds (murmurs, S3 and S4). When using the bell, press lightly. The diaphragm is used for higher-frequency sounds (S1 and S2). A firmer pressure further enhances higher-frequency sounds. We would usually use the diaphragm side.
- Avoid talking into or tapping the diaphragm while you have the stethoscope in your ears. It could cause ear damage!

With the digital stethoscope, measure for about 10 s.

For this project, we have also developed alternative chest pieces that contain piezo-electric elements, these are expected to give cleaner signals as they are not picking up ambient sound.

2.6 ASSIGNMENTS

1. Listen to pre-recorded heart sounds: normal sounds and some pathological cases.
   - Use one of the many sites with training examples, e.g. Fundamentals of Heart Sounds.
   - For best listening, you might need to use headphones.

- Select 4 pathological cases, and challenge each other to see if you can correctly identify them when played in random order.

2. Look at signal waveforms in time and frequency domain.

   - Download 4 samples from the Computing in Cardiology (CinC) challenge: The CirCor DigiScope Phonocardiogram Dataset.
   - Make a plot in time domain.
   - Make a plot in frequency domain.

   Repeat for each signal waveform. Make sure your time axes and frequency axes are correct, and that the plots have titles. Include your results in your midterm report.

Bibliography

[1] D. B. Springer, L. Tarassenko, and G. D. Clifford, "Logistic regression-HSMM-based heart sound segmentation," *IEEE Transactions on Biomedical Engineering*, vol. 63, no. 4, pp. 822–832, 2016.

To access scientific literature, we recommend to look them up in Google Scholar.

Chapter 3

HARDWARE SETUP AND DATA ACQUISITION

Contents

3.1 OVERVIEW

Over the next few weeks, you will be using a variety of audio equipment. This chapter gives a first introduction. Subsequent chapters will guide you on the experiments you will conduct with them.

3.2 SOUNDCARD

For all audio measurements, we will be using the Presonus AudioBox 1818VSL USB sound card, shown in Fig. 3.1. It has 8 analog inputs (ADCs) and 10 analog outputs (DACs). We will use this in combination with a Raspberry Pi 5, shown in Fig 3.2 that is pre-loaded with recording software.

Figure 3.1. Presonus sound card: (a) front, and (b) backside

Figure 3.2. Raspberry Pi 5

3.2.1 Connecting the soundcard

The sound card needs to be powered with a power cable. Plug this cable first into the sound card on the back and then in the wall outlet. Then next, plug the usb cable in the sound card's USB 2.0 port on the back and then in one of the available USB port's on the Raspberry Pi 5.

3.2.2 Connecting the Raspberry Pi

The Raspberry Pi 5 first needs to be connected to its power supply. A USB-C type power supply will be provided with the Raspberry Pi. Connect this to the available USB-C port on the Raspberry Pi 5 and plug the adapter into the wall outlet. Note!: The Raspberry Pi 5 will immediately boot up, which can be verified with a green status light on the device. If this is red, it means the Pi is turned off and the light needs to be pressed to turn the Pi on.

One needs then to connect an Ethernet (RJ-45) cable, shown in Fig. 3.3 to the Raspberry Pi's available 'Ethernet port'. Connect the other side to your computer!

3.2.3 Connecting the microphones

The microphones need to be powered to generate an output signal. This is done by switching on the phantom power on the soundcard. There are two buttons for the phantom power, one for the channels Analog 1-4 and one for Analog 5-8.

Figure 3.3. Ethernet cable with RJ-45 connectors.

Figure 3.4. Rode adapter plug

The soundcard uses XLR plugs. The XLR standard uses a balanced audio interface, which admits long cables. Electromagnetic interference would couple into the two signal wires in the same way, and gets rejected by a differential circuit at the receiving end.

The microphones of the linear 6-channel array and the Piëzo-based microphones' setup use XLR plugs and can be directly plugged in (two in the front, four in the back; check the numbers on the cables to keep them in the right order). However, the stethoscope microphones use 3.5 mm mini-jack plugs and need an adapter. Also, the phantom voltage of the soundcard (12–48 V) is too high for our stethoscope microphones (3–5 V). Use the Rode VXLR+ adapter plug to connect the 3.5 mm mini-jack plugs of the microphones to the soundcard (Fig. 3.4). The jack input is threaded, so you can screw in the plug to secure the connection.

3.2.4 Recording

IP-3 Recording Tool For recording, you will be able to use the "IP-3 Recording Tool" on the Raspberry Pi 5. This is a home developed software tool to mitigate the difficulties that installing and using drivers for this sound card on Windows and MacOS might bring.

In order to use this tool, make sure you've followed the steps before and the Raspberry Pi 5 is connected to your laptop and turned on.

Open up your web browser (Mozilla Firefox, Safari, Google chrome etc.) and browse to **"http://ip3.com"**. If done so, one should see the Home screen of the IP-3 Recording Tool appear. From here, you can follow a tutorial on how to use this software, but all details are available in this manual as well. Once you are familiar with the tool already, you can skip this and go straight to the 'main menu'.

Setting up the Tool Before one can use the IP-3 Recording Tool. You'll need to make sure that it is using the right sound interfaces to make recordings and play sounds. In the 'Settings' menu, you can select the right interface. Make sure to select as an input interface: "

As an output interface, you can select the 'USB audio device'. This is the same USB sound card that is used during the lab courses when connecting two microphones to your laptop.

If done correctly, you should be able to see the correct interfaces on the bottom of the 'Main Menu', when going back.

Make your First Recording In order to make a recording, go to the 'Record Audio' menu in the 'Main Menu'. Here you can specify all the details for the recording, like:

- Recording duration: how long you want to record;
- Sample frequency: on what sample frequency to record;
- Number of Channels: with how many microphones you want to record. The sound card will always start to count from the first channel. So if e.g. 6 channels are selected, then input channels 1, 2, 3, 4, 5 and 6 on the sound card will be used.
- Playback during recording: with this option you can play a .wav file that you upload to the Raspberry Pi 5 simultaneously with your recording. The sound will be played over the selected output interface. So make sure to connect an output amplifier or speakers to the headphone port on this output device.

When you click on 'Start Recording', the recording will immediately start. If playback is turned on, a sound will immediately start to play as well. Do not worry about the length of your uploaded .wav file. If this is too short for your recording, it will automatically be repeated. If it is too long, the software will automatically truncate it to the correct length of your recording.

When the recording is done, they will automatically be saved to the Raspberry Pi's internal storage.

Checking and downloading your recording After making a recording, it could be wise to check if the recording is made well. In the 'Playback' menu, you can listen back your recordings. The sound will be played through your laptop, so do not forget to turn up the volume!

In 'Plots' you can create a quick plot to check your recording. If you want to download these as well, do not forget to click on 'Save Plot'.

Your recordings can be found in the 'Files' menu, from the 'Main Menu'. Here you can download all the recordings, or a selection of your choice. Saved plots can also be downloaded from here.

Do not forget to delete your recordings after you are finished!!!. Other groups might benefit from your work and deleting is not done automatically.

3.2.5 Loopback

For some of the measurements, you will use an active source (loudspeaker). You could either play an existing audio file, or generate one in Python, usually a white noise-like signal. This signal is fed to the loudspeaker, but it is also needed as a reference signal for processing of the measurements (deconvolution). To account for the hardware delay of the sound card as well as potential filter effects, we would not use the clean signal as generated by the software as our reference. Instead, we record the output signal of the amplifier by connecting an RCA to 3.5 mm cable with a 3.5 mm to 6.35 mm Jack converter to the 'recording output' on the amplifier and analog input channel 7 or channel 8 of the sound card.

3.3 LINEAR MICROPHONE ARRAY

The linear microphone array is a construction shown in Fig. 3.5. It has two rows which each accept $M = 6$ Behringer microphones. The spacing for the top row is $d = 5$ cm, and for the bottom row it is $d = 10$ cm.

The microphones are connected to the data recording using long cables. Pay attention to the correct ordering of the cables, at both ends!

The microphone array is placed at a fixed distance from a wall, and one or two sources (loudspeakers) produce sound. You will need to rotate the array and measure the angle the array makes to each of the sources. For this, use a protractor (NL: gradenboog), or the compass of your mobile phone. This gives an accuracy of about 1°.

3.4 PHANTOM HEART MODEL

To test your algorithms, we will use a phantom model of the heart (Fig. 3.6). It is a wooden box containing two hidden loudspeakers, and 7.5/15 cm of foam on top. You can remove the foam and measure the exact locations of the holes ("valves") to obtain ground truth for your localization algorithms.

Test signals will be played on the loudspeakers using the headphone output on the USB Audio Device. However, these outputs first need to be amplified. We will use a generic (albeit a bit oversized) amplifier for this: the Monacor SA-230/SW "universal stereo mixing amplifier" (Fig. 3.7). You will need to connect the two speakers to the "right" and "left" speaker clips at the back. Connect the USB Audio Device output to the L and R Aux input using a 3.5 mm jack to RCA cable.

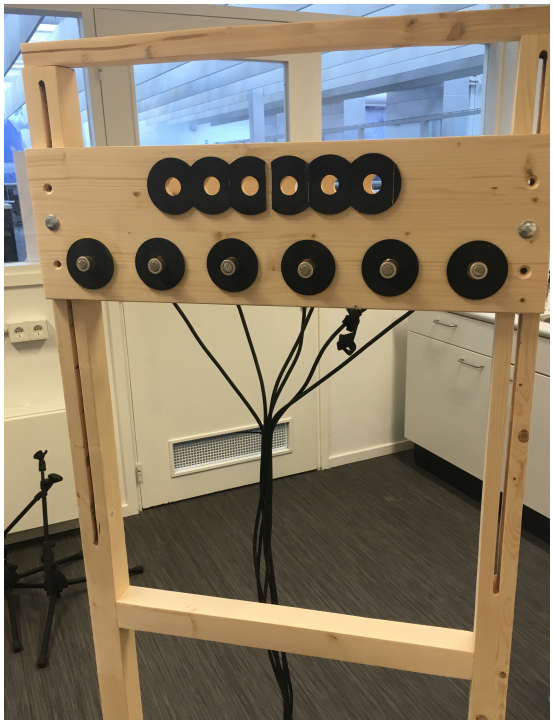


Figure 3.5. Linear microphone array.

(*a*) 

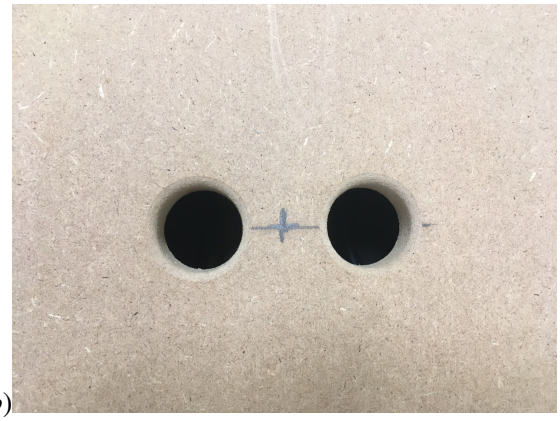
(*b*) 

Figure 3.6. (*a*) Phantom heart model, with (*b*) the hidden “valves”.

Figure 3.7. Output amplifier

(*a*)  (*b*) 

Figure 3.8. (*a*) Stethoscope array; (*b*) Lavalier microphone

3.5 STETHOSCOPE ARRAY

On the phantom (and possibly a human volunteer), we will measure signals using a 6-element stethoscope array, shown in Fig. 3.8. Each element is composed of a regular stethoscope "head", to which we fitted a Lavalier Clip Speech microphone (Relacart LM-P01); frequency range: 20–18000 Hz).

The custom-made array is designed to have some flexibility; the element spacing is 5 cm.

Chapter **4**

M1: SINGLE CHANNEL SIGNAL ANALYSIS

Contents

In this module, we look at PCG recordings acquired with our digital stethoscope. We consider just a single microphone, and look at preprocessing steps and basic analysis. Our goal is to preprocess and segment the data into individual heart beats, and annotate the S1 and S2 intervals.

Learning objectives The following is learned and practiced in this module:

- The analysis of a recorded PCG signal
- The segmentation of this signal into S1 and S2 intervals

Preparation

- Read this module
- Review the background knowledge in Ch. 1, in particular the preprocessing functions
- Make sure your Python IDE is ready to go!

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- On Brightspace: pre-recorded audio recordings.

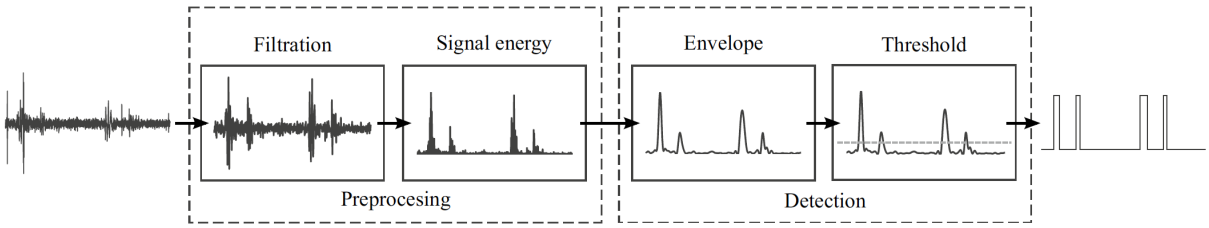


Figure 4.1. PCG segmentation steps [1]

4.1 OVERVIEW

The generic steps to come to a PCG segmentation are shown in Fig. 4.1. After acquiring the data, we first apply a bandpass filter to select the frequency range of interest. Since our data rates are initially much higher than needed, we subsequently apply downsampling (not shown in the figure). Next, we construct the instantaneous energy signal (either by squaring or by using the Shannon energy function), and detect the envelope. Using thresholding or more advanced peak finding algorithms, we can detect the S1 and S2 events, and segment the signal into individual heart beats. This is the hardest part, in particular if you wish to be robust against interfering signals and pathological cases.

It is best to select a "standard" sample rate at which we will do all our data processing; let's set this at $F_s = 4$ kHz (a convenient number if the initial sampling rate was 48 kHz).

We provide the processing steps in this Module in the form of cookbook instructions. Once you understand the recipe, you are of course free to deviate!

4.2 LOWPASS FILTERING

In biomedical signal processing literature, often Butterworth filters are used, perhaps because of tradition, but also because the passband frequency response is maximally flat. Inspecting and interpreting the shape of pulses is very important, and the filters should not introduce distortion.

Once it is known that a signal of interest has a certain frequency content (e.g., it is mostly below 1000 Hz), then a lowpass filter can be applied to filter out the higher frequencies and thereby reduce the noise. Very often, also a DC filter is applied to remove very low frequencies, which are seen in the recordings as "baseline wander". E.g., components below 3 Hz or 5 Hz are frequently nulled.

Although Butterworth filters have a maximally flat amplitude response, their phase response is not ideal. An ideal filter looks like a simple delay in the passband, i.e., it just delays the desired signals. In frequency domain, a delay response is $H(\Omega) = e^{-j\Omega\tau}$. Thus, the phase is linear (negative slope corresponding to τ). If the phase is not linear, then some frequency components appear earlier or later than others, leading to pulse distortion. For Butterworth filters, this occurs in particular at the higher frequencies in the passband.

To obtain a linear phase, the filter impulse response $h(t)$ must be symmetric: $h(t) = h(T-t)$, for some T. Here, $T/2$ is the filter delay. Thus, if we require causality, only FIR filters can have linear phase. However, if we process recorded signals, we don't need to require causality.

To achieve a zero phase, biomedical signal processing often uses the following trick: apply the filter twice, but once backwards:

$$g(t) = h(t) * h(-t)\,.$$

The frequency response is

$$G(\Omega) = H(\Omega) \cdot H^*(\Omega) = |H(\Omega)|^2\,.$$

Thus, we could design a Butterworth filter $h(t)$ with a damping of 20 dB at 1000 Hz. When applying the filter twice (once backwards), we have a noncausal lowpass filter which is maximally flat, has zero phase, and a damping of 40 dB at 1000 Hz. Of course, also the filter order doubled.

In Python, you can use the command (from the SciPy package)

```
from scipy import signal
b, a = signal.butter(order, [low, high],  btype='band')
filtered_data = signal.filtfilt(b, a, data)
```

low & *high* are calculated by dividing the low or high frequency by the nyquist frequency.
Be aware of transient effects at both edges of the signal.

4.2.1 Assignments

1. From Brightspace page of the course, download prerecorded human signals. Note the sample rate.
2. Construct a 2nd order Butterworth bandpass filter $h(t)$ with cutoff frequencies at 10 Hz and 800 Hz. Then construct the zero-phase (forward-backward) filter $g(t)$. Make plots of the time-domain and frequency-domain responses of this filter. How much damping do you have at 2000 Hz?

   Note that, to find $g(t)$ and to perform zero-phased filtering, you don't need to calculate $h(t)$ but can directly perform forward-backward filtering on an impulse input (dirac delta pulse) using `g = filtfilt(b, a, impulse)`.
3. Perform zero-phased filtering on your recorded signals. Make plots of the signal before and after applying the filter. It is important to note if the main features of the desired signal (e.g., location of peaks) are preserved, this also indicates if the filter is truly zero-phase. Is this indeed the case? Why is it important to preserve the location of the peaks for later use in Module 3 & 4?

4.3 DOWNSAMPLING

When sampling signals, you might be tempted to use a sample rate F_s that is as high as possible. Indeed, this might enable you to accurately spot the location of important spikes or events. E.g., for audio signals

it should be more than sufficient to sample at 48 kHz, but high-end equipment often oversamples by a factor of 4 to 192 kHz.

This has two disadvantages:

- The higher frequencies have no useful signal; you just collect noise;
- Computational complexity and memory storage is unnecessarily high.

Thus, if you inspect your sampled signal in frequency domain and discover that it actually is bandlimited and occupies only frequencies below some value, you might want to resample it to a lower frequency. This is called decimation or downsampling. To do this, you need two steps:

- You need to satisfy the Nyquist condition after downsampling. E.g., if the new sample rate is 4 kHz, then you first need to apply a (digital) lowpass filter that cuts off at 2 kHz.
- Next, downsampling by a factor M is implemented simply by keeping only every Mth sample:

```
x_new = x[::M]
```

4.3.1 Assignments

1. Calculate the downsampling factor M needed to obtain a new sample rate of $F_s = 4$ kHz. Apply downsampling by this factor to the filtered signal from previous section.

   Note: to avoid aliasing, your signal needs to satisfy the Nyquist rate condition at the new sample rate. This is why it is important to have sufficient damping at $F_s/2$. If you consider that the damping (calculated earlier in this section) is not sufficient, increase the filter order or reduce the cut-off frequency. Document your results and report the final values used.

2. Make plots of the resulting signals.

   - What can you say about the signal quality?
   - If you consider that the noise level is too high, would it help to reduce the order or the band of the bandpass filter?

4.4 SPECTROGRAM ANALYSIS

Once you have acquired a signal, it is a good idea to always inspect it (i.e., plot it), both in time domain and in frequency domain. You want to avoid problems due to clipping, too low gain setting, too much noise, or interference.

In the EE2S1 lab sessions, you have also plotted time-frequency spectrograms. You did that by splitting the input signal into short (possibly overlapping) segments, and applying the FFT to each segment.

In Python, there are several tools to automate this for you. You can use them once you are familiar with the processing and know what to expect. A straightforward tool (but "legacy" in lieu of the `ShortTimeFFT` tool) is `signal.spectrogram`. An example is

```
from scipy.signal import spectrogram
f, t, Sxx = spectrogram(signal, Fs, nperseg=256, noverlap=128)
```

In this example, the signal is segmented in blocks of 256 samples; on each of these an FFT is performed. This gives a time-frequency spectrogram with 256 samples in frequency domain. In this example, `noverlap` is half the block size; this means that the next block overlaps half with the previous one. The output parameters `t` and `f` are vectors that define the time axis and frequency axis; you can use them when plotting the spectrogram `Sxx`:

```
plt.imshow(t,f,Sxx)
```

You could choose to plot the spectrum in dB, using

```
Sx_dB = 10 * np.log10(Sxx)
```

A more fancy and recent tool with similar results is `signal.ShortTimeFFT`. It uses a lot of parameters, most of which will not make sense to you. E.g., it uses a "window" and a "hop". When you did the spectrogram by hand, you used in fact a rectangular ("boxcar") window, and a hop equal to the length of the window. Instead, you could use a triangular window or a Hamming window, and a hop equal to half the window size or even smaller. See `signal.get_window`. The purpose of the window is to have a better frequency response. E.g., for a sinusoid, you would expect to see a spike in frequency domain, but with a rectangular window, it becomes a sinc function. Different window functions reduce the side lobes of the sinc, at the expense of a wider main lobe (some loss of resolution). Here is a simple example in Python:

```
from scipy.signal import ShortTimeFFT, spectrogram
from scipy.signal.windows import gaussian

N = len(signal)
win = ('gaussian', 1e-2 * fs)  # Gaussian with 0.01 s standard dev.
SFT = ShortTimeFFT.from_window(win, fs, nperseg=256, noverlap=125,
                fft_mode='centered', scale_to='psd', phase_shift=None)

Sx2 = SFT.spectrogram(signal)
t_lo, t_hi = SFT.extent(N)[:2]  # time range of plot
Sx_dB = 10 * np.log10(np.fmax(Sx2, 1e-4))
plt.imshow(Sx_dB, origin='lower', aspect='auto', extent=SFT.extent(N))
```

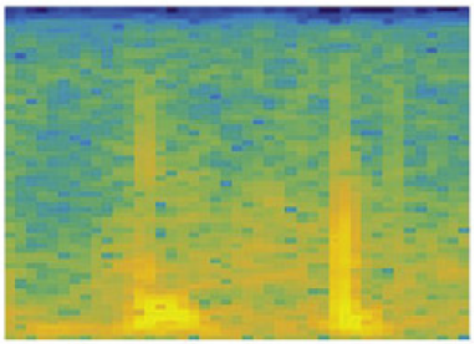


Figure 4.2. Spectrogram of 1 heartbeat cycle

4.4.1 Assignments

1. Make a spectrogram. Make sure to correctly label the time and frequency axes.

    - Compare to the time-domain plot. If our goal is to segment the signal, does the spectrogram give additional information?

The result could look like Fig. 4.2. In this figure, a log-scale (dB) has been used to better see the weak amplitudes.

4.5 ENERGY DETECTION

4.5.1 Amplitude normalization

For many algorithms that use a threshold, it is convenient if the amplitude of the input signal is first normalized. A simple technique is

$$x_n(t) = \frac{x(t)}{\max|x(t)|}$$

This normalizes the range of $x(t)$ to the interval $[-1, 1]$.

4.5.2 Shannon energy

If $x(t)$ is a signal, then we could call $x^2(t)$ the "instantaneous energy". If we integrate it, we find the energy.

It is found that this function ignores the smaller amplitudes and emphasizes the peak amplitudes. If we would like to focus on "medium" amplitudes, people have proposed to use the Shannon energy,

$$E(t) = -x_n^2(t)\log\left(x_n^2(t)\right).$$

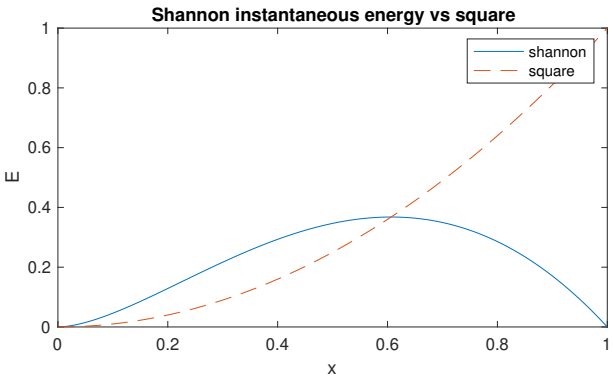


Figure 4.3. Shannon energy function

See Fig. 4.3. It is assumed that the input signal has been normalized.

$E(t)$ is usually a peaky signal. To find the Shannon Energy Envelope (SEE), we can use a lowpass filter on $E(t)$. A typical cut-off frequency in our context would be 10 to 20 Hz. Fig. 4.4 shows an example of what you can expect to see at this point.

4.5.3 Assignments

1. Normalize the signal amplitude of your down-sampled signal in Section 4.3.1 to fall within the range $[-1, 1]$.
2. Construct the Shannon energy signal of your normalized signal.
3. Construct the Shannon energy envelope (SEE).
    - To preserve the timing, do you need to use a zero-phase filter (`filtfilt`)?
    - Make plots and document your results.
    - The result is expected to be sufficiently smooth and show peaks for the S1 and S2 signals. Is that indeed the case?

4.6 SEGMENTATION

We have reached the final step in the processing chain, and also the hardest. From the SEE, can we detect the S1 and S2 intervals?

Cardiac signals are quasi-periodic (i.e., not exactly periodic), and in many cases we would first want to segment the signal into intervals containing a single beat. We would usually develop algorithms based on the selection of peaks. While you probably have used `argmax(abs(x))` before, a much more powerful command is

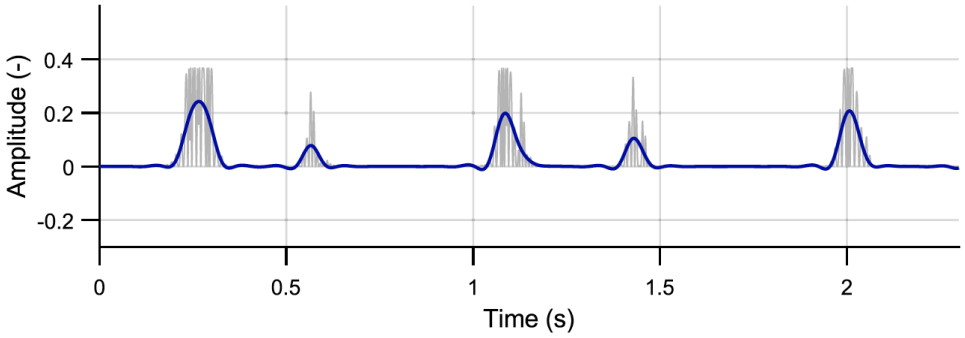


Figure 4.4. Shannon energy (gray) and its envelope (blue) [1]

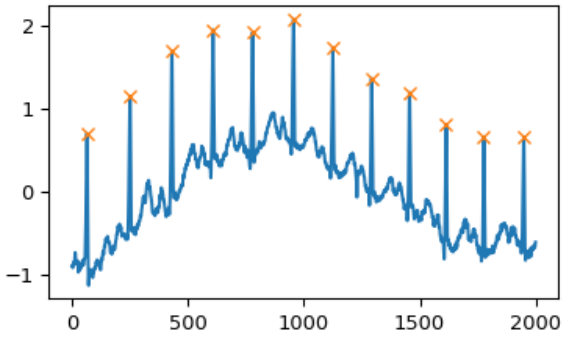


Figure 4.5. ECG signal, and detected peaks using “distance = 150” (SciPy Documentation)

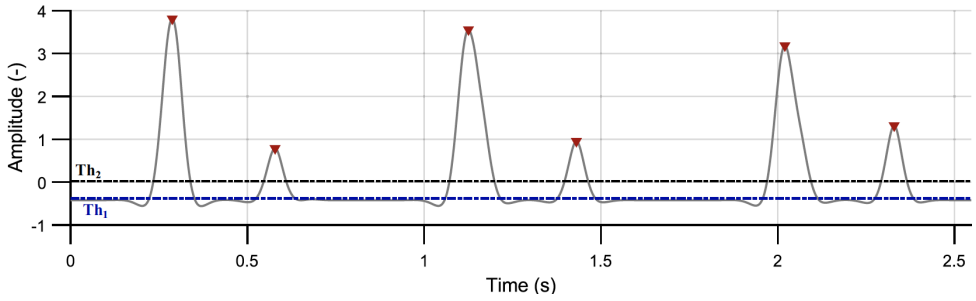


Figure 4.6. Shannon energy envelope, and the detected peaks above a threshold [1]

```
peaks, _ = signal.find_peaks(x, height, distance)
```

The `find_peaks` function allows you to find a list of local maxima. It has many options. E.g., you can specify that your peaks need to have a certain distance to each other, set a minimum height, etc. Fig. 4.5 shows an example.

1. Normalize the SEE signal. You could either scale to set the maximum equal to 1, or (as in literature) scale such that the standard deviation is 1. In either case, the idea is that if we select detection thresholds, they become scale-independent.

   Note: In literature, you might see that people also subtract the mean, however, we would advice against this, since the signal pdf is not symmetric. It does not make the selection of the detection threshold easier.

2. Use `find_peaks` to select peaks corresponding to the S1 and S2 signal. Use your knowledge on the shape of the signals to guide you (e.g., what is the expected distance between S1 and S2?). You can also define a minimum height and require that the peak is larger than that (e.g., 0.1).

   Fig. 4.6 shows an example.

3. Determine which peak is S1 and which is S2. Note: you cannot base yourself on the amplitude of the peak, as it would depend on the measurement location if S1 is stronger or S2. Instead, consider that the systole (period from S1 to S2) is shorter than the diastole.

   - Show plots with the detected S1 and S2 peaks marked.

4. Determine if your method is sufficiently robust. E.g., S1 and S2 peaks should alternate, but it could happen that one of the peaks is not detected, or an extra peak was detected (in particular in the case of pathology). You could try to make your algorithm more robust. This is a topic of research even today; many options are available.

   - Before making any modifications, what is your classification accuracy? (i.e., how many of the peaks are correctly classified, using your "professional eye" as ground truth?) In literature,

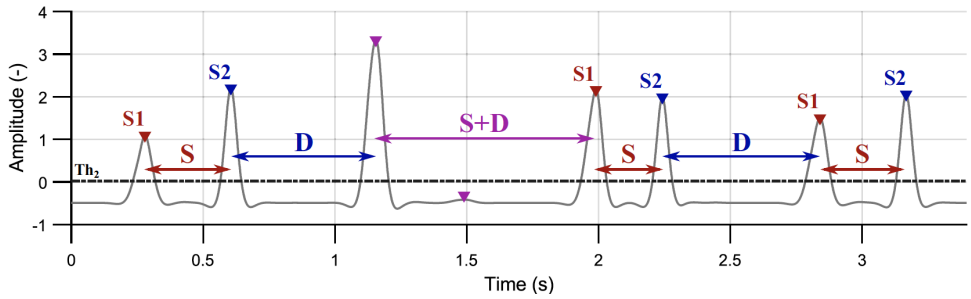


Figure 4.7. Listing of systole and diastole intervals on the SEE, including some processing that resolved a missing peak [1]

accuracies around 75% are reported, but this number depends on the class of signals that are considered, and on the recording systems.

- After inserting improvements, what accuracy are you able to reach?
- Instead of aiming for 100% accuracy, are you able to *flag* intervals where something is wrong? During subsequent processing, we would be able to drop such time intervals from further consideration. This is very hard to automate so this can be done by visual inspection.

Fig. 4.7 shows an example classification at this point, where also a missing S2 peak has been resolved.

5. Around each S1 and S2 peak, select intervals such that we capture the entire S1 and S2 signal. This could be based on a suitable threshold on the SEE.

4.7 INTEGRATION WITH OTHER MODULES

Your teammates working on Modules 3 and 4 will rely on your output to localize the heart valves. You will receive as input a 6-channel heart recording. However, what your teammates actually need for localization are **two separate sets of 6-channel recordings**:

1. One set containing **only S1 segments** (S2 and silent parts removed).
2. One set containing **only S2 segments** (S1 and silent parts removed).

It is not enough to simply provide the start and end times of S1 or S2 segments. Instead, for each segment type, you need to create a **new signal** consisting only of the identified segments, concatenated in order. Take a moment to visualize this process; if anything is unclear, discuss your interpretation with a TA.

Currently, your algorithm focuses on a single signal, but all 6 channels must be segmented simultaneously to preserve inter-channel delays. You need to decide:

- Will you perform segmentation all channels and then find overlapping segments?
- Or will you select the channel with the best quality and base the segmentation on that?
- How will you measure the quality of each channel?

Keep in mind that some channels may have poor recordings that should be discarded later.

Discuss these decisions with your team to clarify exactly what you need to provide. Although we performed filtering, low-pass filtering, and downsampling in this section, these steps were primarily for segmentation. Your teammates may request the **segmented raw data** without any additional filtering or preprocessing.

4.7.1 Assignments

Your 6 channel segmentation algorithms will later integrate with Modules 3 and 4. Create a function. Create a function `Segmentation(.)` that accepts appropriate inputs (you decide which) and outputs 2, six-channel recordings including S1 and S2 segments each. Use the 6 channel human recordings provided on Brightspace.

4.8 EXTENSIONS

Once the main task is complete, you can consider the following extensions. Choose the one that interests you the most, the grade will depend on **quality**, not quantity, of the work. You are also welcome to propose your own extension ideas, but please discuss and confirm them with us before starting.

Algorithm for automatic noise and artifact rejection:
Design an algorithm to automatically reject noise and artifacts in the 6-channel microphone array. If noise or artifacts affect only a single channel (e.g., due to a microphone or soundcard malfunction), it is better to discard that channel and use the remaining channels for localization. If noise is present across all channels, or if S1 or S2 segmentation is incorrect, discard the segment across all channels. Consider implementing a GUI to facilitate result inspection, especially when manual verification or adjustments are needed.

Improved S1 and S2 segmentation:
Enhance your S1 and S2 segmentation using a labeled dataset with ground-truth annotations, such as the one available on Kaggle Heartbeat Sound. You can evaluate your algorithm's performance with a **confusion matrix**, which measures four key parameters. Depending on your application, some parameters may be more important than others—discuss your interpretation and assumptions with the instructor or TAs before drawing conclusions.

GUI for Segmentation:
Develop a graphical user interface (GUI) for segmentation where you can easily adjust parameters and visualize the final S1 and S2 segments. This allows for quick visual inspection and the option to reject inaccurate segments.

Bibliography

[1] R. Jaros, J.Koutny, M. Ladrova, and R. Martinek, "Novel phonocardiography system for heartbeat detection from various locations," *Scientific Reports*, vol. 13, Sep. 2023.

Chapter **5**

M2: HEART SOUND MODELING

Contents

In this module, we develop a Python algorithm to generate sufficiently realistic S1 and S2 sounds. These will be used later to test the localization algorithms developed in Modules 3 and 4. Further, since these algorithms measure propagation delays, we need the speed of sound to convert that into a location. In this module, we will therefore also estimate the speed of sound in the human chest.

Learning objectives The following is learned and practiced in this module:

- The construction of a simple data model to generate S1 and S2 sounds
- The testing of this model on your S1/S2 segmentation algorithms
- Measuring and estimation of the sound speed in the human chest

Preparation

- Read this module
- Revisit EE2S1 Signals&Systems: Chaparro Ch. 3.3 “The one-sided Laplace transform”

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- Your preprocessed data, and segmentation algorithms from Module 1
- Data recorder
- An actuator (small loudspeaker) and a digital stethoscope

5.1 OVERVIEW

When developing signal processing algorithms, it is tempting to immediately test them on real data, especially when that data is readily available. However, that is not always the best idea. Without access to ground truth, we can only qualitatively inspect if the algorithm seems to do the right thing. We do not have a performance measure. More importantly, most signal processing algorithms are developed based on certain assumptions on the data, e.g., we try to estimate a certain frequency, or a delay, or some other parameter. That means that we have an underlying *parametric model* on which the algorithm was developed.

It makes sense to make that model more explicit. In signal processing, data models are used to generate sufficiently realistic but not overly complex simulated data. Usually, there are just a few parameters to set, e.g., an amplitude, a frequency, a location. Everything that is not captured (the model error) is considered to be part of "the noise". Initialy we develop and test our algorithms on the synthetic data. The advantage is that we have ground truth (the parameter settings), and can vary the noise level. In this way, we can statistically assess the performance of the algorithms. In future MSc courses on Detection and Estimation, we show lower bounds on how well we could estimate the unknown parameters, as function of the model, the number of samples, and the noise level.

In our application, we would like to test the heart sound localization algorithm on a phantom setup, where we play heart sounds over two loudspeakers. While we have many heart sound data files, we do not have real data for each heart valve separately. This gives another motivation to try and generate such synthetic data ourselves.

In literature, models for the individual components of each heart sound are not readily available. It is hard to individually measure and characterize the individual valve signals, due to their temporal and spectral overlap. A good model for generating reliable synthetic PCG must consider the following points [1]. First, the PCG model must be able to generate transient signals having low-frequency components since the main energy of heart sound spectrum is in the range of 20–200 Hz. Secondly, the PCG model must take into account asynchrony between valve closures that causes the splitting of heart sounds. Finally, beat-to-beat durations vary and therefore it is important to incorporate the HRV in the PCG model.

In this chapter, you will develop a sufficiently simple model for heart sound signals. We will model the four valves separately. During System Integration, this model will be used to test the localization algorithms developed in Modules 3 and 4 to simulated data. After that works, you will be ready to test the algorithms on a phantom setup, where your generated sound signals are played over loudspeakers hidden inside a box filled with foam. If all that works, we can go to real data.

5.2 SINGLE VALVE MODEL

The heartbeat sound occurs when a valve is closing and the blood flow is suddenly blocked; this produces an impulse which causes the membrane of the valve to vibrate as in a percussion drum. There could be multiple resonance frequencies. Even in an ideal, circular drum, multiple modes (standing waves) are

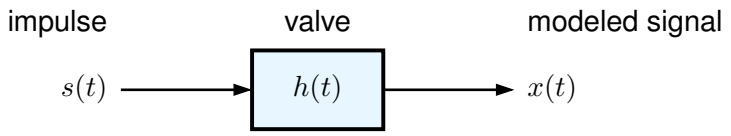


Figure 5.1. Model of a valve as an LTI system

possible, each with their own resonance frequency. However, unlike a string (like on a guitar), where the different frequencies of vibration are mathematically related in a simple, predictable way (called "harmonics"), the drum's vibrations don't follow such a simple relationship. So, the different frequencies that the drum produces aren't connected in an easy-to-predict pattern. This makes the sound from a drum more complex and less predictable than the sound from a vibrating string. The situation becomes more complex if the membrane is over a closed container that can vibrate itself as well.

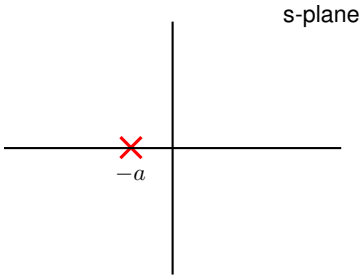
As often in signal processing, we will therefore not attempt to make a physically realistic model of the sound which would be really hard. So instead, we focus on simpler methods to model the sound. Let us model the "valve system" simplistically as an LTI system with impulse response $h(t)$, driven by an input $s(t)$ which is an impulse, $s(t) = \delta(t)$; see Fig. 5.1. We will model $h(t)$ as a sum of poles.

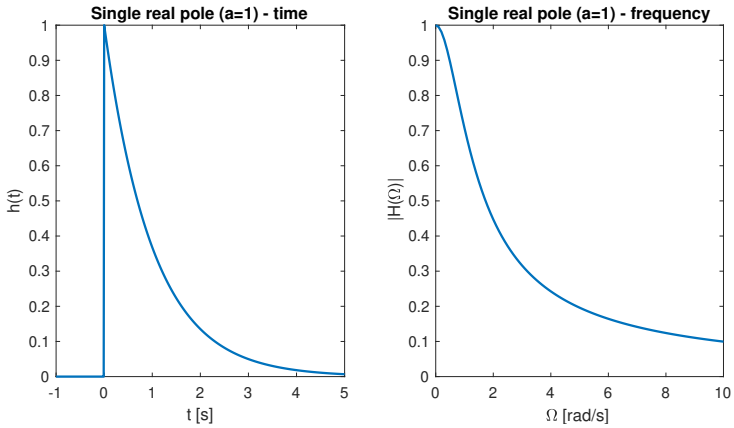
5.2.1 LTI systems for signal modeling

In signal processing, particularly when modeling natural phenomena like heart sounds, we often need to generate signals that start strong and then decay over time—transient signals. A key tool for creating such signals is the concept of Linear Time-Invariant (LTI) systems. LTI systems provide a structured approach to modeling how signals evolve, as they describe how an input signal is transformed by the system's properties over time. The power of LTI systems lies in their simplicity and versatility, allowing us to break down complex signals into well-understood components. For our purposes, we need to build a signal that mimics the behavior of a heart valve closing, which requires generating a sound that starts with an initial burst and gradually fades out—a typical transient response.

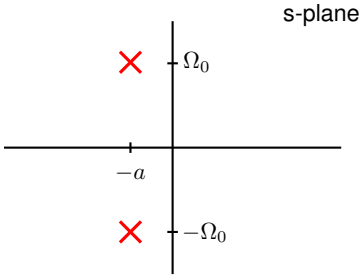
Step 1: Generating a transient signal with decay To capture this decaying nature of a transient signal, we can start by modeling the response using an exponential decay. The exponential function naturally represents signals that diminish over time, which is crucial when modeling the fading vibrations of a heart valve after it closes. From EE2S1 Signals&Systems, you know that an exponential decay is generated by an LTI system with a single pole in the system's transfer function, see Fig. 5.2(a).

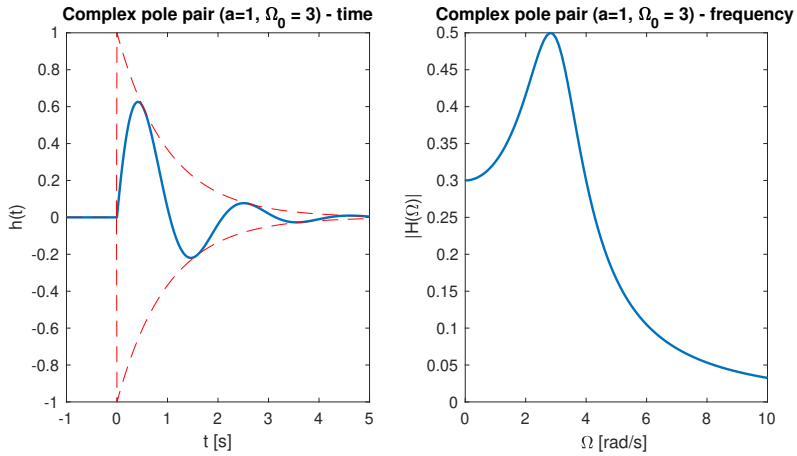
Step 2: Introducing oscillation While the exponential decay captures the gradual dying out of the signal, heart sounds are not purely smooth decays; they involve oscillations due to the vibrations of the valve membranes. To introduce these oscillations, we add a sine wave component to the model. Sine waves naturally represent periodic behavior and allow us to capture the frequency content of the signal. By combining the exponential decay and the sine wave, we model a signal that starts with an initial burst



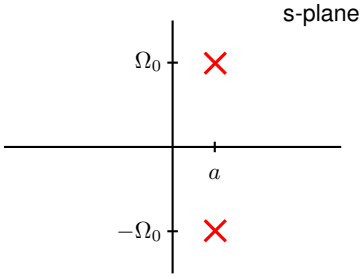


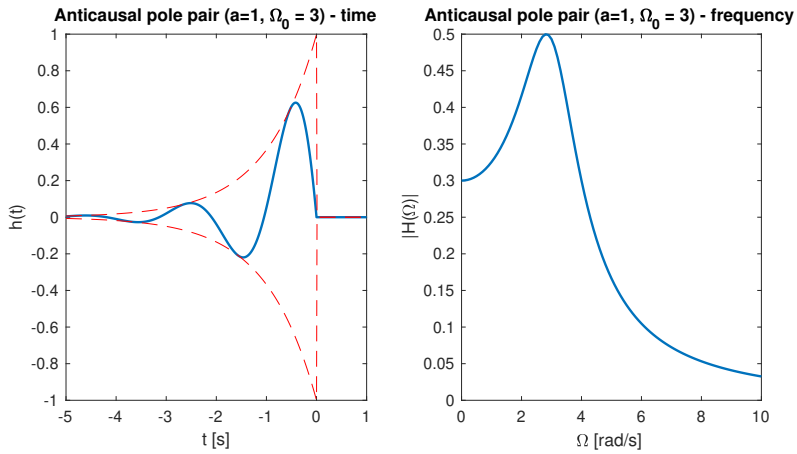
(a) Single real pole





(b) Single real pole





(c) Complex pole pair in the RHS

Figure 5.2. LTI systems: connections between pole locations, impulse responses, and frequency responses.

and oscillates while gradually fading out, closely mimicking the dynamics of heart sounds, see Fig. 5.2(b).

Step 3: Ensuring continuity When modeling such signals, it's important that they begin from zero at the point of closure, representing the moment the valve shuts. This is where the concept of a zero in the LTI system comes in. A zero in the system manipulates the phase, and can ensures that the signal starts from zero, rather than some non-zero initial condition, ensuring continuity in the onset of the transient response. (Unfortunately, we don't have a simple relation between the zero location and the initial value of the response.)

Step 4: Capturing initial vibrations To further refine our model, we can introduce poles in specific locations of the system. A pole on the right-hand side (RHS) of the complex s-plane results in an anti-causal impulse response.[1] This can model an initial burst of energy in the signal, representing the first vibrations when the valve closes. This initial vibration mimics the real-world behavior of the valve, which experiences a brief, intense movement before the vibrations dampen out over time. Other LHS poles help model the rapid decay of energy after the initial burst, allowing the sound to fade out smoothly. As the oscillations die out, the signal settles into a quieter state, much like the natural dampening of vibrations in a physical object, see Fig. 5.2(c).

By combining these elements—exponential decay for fading, sine waves for oscillation, zeros for starting from rest, and RHS poles to capture the initial burst of energy—we can construct a model for transient signals that closely resembles real-world phenomena, like heart valve sounds. LTI systems provide a powerful framework for building such models, giving us the ability to control the signal's behavior with precision.

5.2.2 LTI system theory – recap

In EE2S1 Signals&Systems, when we studied the Laplace transform, you have seen examples of the responses of poles. Let's look at some options.

Single real pole

$$h(t) = e^{-at}u(t) \quad \Leftrightarrow \quad H(s) = \frac{1}{s+a} \qquad (a > 0; \text{ROC:}\{s > -a\})\,.$$

For a single real pole with location $s = -a$, the time domain response is a real exponential. The frequency domain amplitude response is derived from $H(s)$ by setting $s = j\Omega$, i.e., $|H(j\Omega)|$. It will peak close to the pole, i.e. at $\Omega = 0$. See Fig. 5.2(a).

In time domain, for $t = 1/a$, the amplitude is reduced to $e^{-1} \approx 0.37$. Thus, $1/a$ determines the duration of the pulse. This property is often used to estimate a (cf. the "RC time" in circuit theory).

[1] It is assumed that the $j\Omega$ axis is in the ROC, to ensure stability.

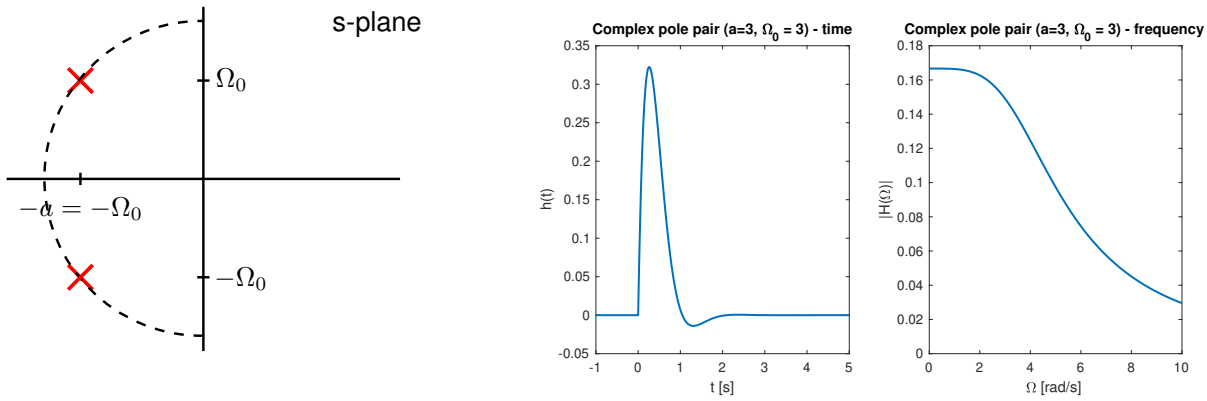


Figure 5.3. Complex pole pair ($a = \Omega_0$): 2nd order Butterworth

Complex pole pair

$$h(t) = e^{-at}\sin(\Omega_0 t)u(t) \quad \Leftrightarrow \quad H(s) = \frac{\Omega_0}{(s+a)^2+\Omega_0^2}, \qquad (a>0;\ \text{ROC:}\{s>-a\}) \tag{5.1}$$

For a complex pole pair with location $s = -a \pm j\Omega_0$, the time domain response is a damped sinusoid. The frequency of the sinusoid is Ω_0. See Fig. 5.2(b). If $a = 0$, there is no damping and the sinusoid will go on forever. For large a, the damping is strong and not many wiggles are seen in time domain. In frequency domain, Ω_0 is seen as the location of the peak. Without damping (poles on the imaginary axis), the peak will be very high. With strong damping, the peak is suppressed. E.g., with $a = \Omega_0$, we obtain a 2nd order Butterworth filter, and the peak is gone (see Fig. 5.3). The time-domain shows only a single wiggle. Thus, Ω_0/a determines the sharpness of the peak (the "quality factor").

Effect of a zero

$$h(t) = e^{-at}\cos(\Omega_0 t+\theta)u(t) \quad \Leftrightarrow \quad H(s) = \frac{e^{j\theta}}{s+a-j\Omega_0} + \frac{e^{-j\theta}}{s+a+j\Omega_0} = \frac{(s+a)\cos(\theta)-\Omega_0\sin(\theta)}{(s+a)^2+\Omega_0^2}$$

If we have a complex pole pair and a single zero, then that zero controls the phase of the sinusoid. The exact expression for the zero seems rather involved, but it is not really of interest. In the earlier figures, the phase θ is seen at the start of the response ($t = 0$), in this case it determines whether the response starts with a jump (due to the step $u(t)$) or is continuous.

In the frequency domain, a zero close to the imaginary axis will pull down the amplitude response. E.g., a zero at $s = 0$ causes the response to be 0 at $\Omega = 0$. The preprocessing bandpass filter that we applied had a zero at $s = 0$.

Delay A delay by τ is inserted by multiplying $H(s)$ with $e^{-s\tau}$. This has no effect on the amplitude spectrum. This function does not add poles or zeros.

Table 5.1. Example values for the valve parameters [2]

	M	**T**	**A**	**P**
Duration (ms)	20	20	20	20
Frequency (Hz)	50	150	50	30
Rel. amplitude	1	0.5	0.5	0.4
Onset delay (ms)	10	40	300	330

Poles in the RHS: anti-causal response

$$h(t) = e^{at}\sin(-\Omega_0 t)u(-t) \quad \Leftrightarrow \quad H(s) = \frac{\Omega_0}{(s-a)^2+\Omega_0^2} \qquad (a>0;\,\text{ROC:}\{s<a\})\,.$$

Poles in the right-hand side (RHS) of the s-plane can be used to model anti-causal (but stable) signals. This corresponds to the choice to include the $j\Omega$-axis in the ROC. Looking at the definition of the Laplace transform, you can observe that time reversal ($t \to -t$) is equivalent to $s \to -s$. See Fig. 5.2 (c). This could be used to model the onset of a pulse. Cf. your SEE plots where you probably observe that the energy of a pulse gradually builds up over a period of 15 ms.

Note by comparing to Fig. 5.2(b) that the amplitude response is unchanged.

Discrete-time domain We work on a computer, and therefore all our signals are sampled. We can easily transform $h(t)$ and $H(s)$ to the sampled domain by the "method of impulse invariance" (as seen in EE2S1), i.e.,

$$h[n] = h(nT_s), \qquad n = 0, 1, \cdots$$

Equivalently, we obtain $H(z)$ from $H(s)$ by the relation $z = e^{sT_s}$. We have seen in EE2S1 that the effect of this transformation on the poles is: if s_k is a pole of $H(s)$, then

$$p_k = e^{s_k T_s}$$

is a pole of $H(z)$. Similarly for the zeros. More specifically, if $s_k = -a + j\Omega_0$, then $p_k = re^{j\omega_0}$, with $r = e^{-aT_s}$ and $\omega_0 = \Omega_0 T_s$. If $e^{-s\tau}$ is a delay in $H(s)$, then this becomes z^{-n}, with $n = \tau/T_s$ (suitably rounded).

Instead of designing $h(t)$ and then transforming it, we might as well design $h[n]$ directly. When making plots of $h[n]$, we would denote the time axis in seconds, using $t = nT_s$. Similarly, when making frequency-domain plots, we would denote the frequency axis in Hertz, using $F = F_s\,\omega/(2\pi)$, such that $\omega = 2\pi$ corresponds to $F = F_s$.

Once we have $H(z)$ (in particular, its poles, zeros and delays), then we can readily construct $h[n]$ in Python.

5.2.3 Suggested values

We could model each valve as a single pole pair. In that case, each valve has 4 parameters: the location of the pole (which determines the frequency and the duration), the amplitude, and a time offset. A zero could be placed such that the response does not start with a jump. Alternatively, a noncausal onset could be modeled with a pole pair in the RHS; you will have to match amplitudes and phases such that the result is continuous at $t = 0$. An onset delay can be used to guarantee causality.

Because no two heartbeats are the same, and the values used rely on the model, a variety of "typical values" is found in literature. The values in Table 5.1 were mentioned in some resources but have been adapted to our single pair pole and zero model.

In [1], it is mentioned that "the mitral component (M1) is slightly higher in intensity and frequency than the tricuspid component (T1). These two components are normally 20–30 ms apart in normal subjects. This time delay, called the time split of S1, is caused by the asynchronous closure of M1 and T1. The aortic component (A2) is louder than the pulmonary component (P2) and these two components are normally less than 30 ms apart during exhalation in normal subjects, rising to around 50–60 ms apart at the end of inhalation." Thus, this would suggest slightly different values in Table 5.1. The model in [3] uses an aortic component of duration 60 ms, with an onset of about 15 ms and a decay period of 45 ms.

Note that the relative amplitudes really depend on the location of the measurements (i.e., the distance to each valve), and this location is often not mentioned.

Literature suggests that 3–6 pole pairs are sufficient to model S2 (two valves); the parameter values vary from beat to beat. In literature, at some point chirps were suggested to model the wider frequency content seen in measurements. Physically, this could be motivated by the increasing pressure during a beat, which might increase the tension of the modeled "drum" which increases its pitch. However, in time-frequency plots, no clear evidence was found to motivate a chirp (the resolution in such plots is insufficient to make strong conclusions).

In combination with chirps, elaborate but unmotivated amplitude functions were introduced to model the onset of a pulse [3]. As alternative, we suggest to use a pole pair in the RHS (there is no literature on this).

5.2.4 Assignments

1. In Python, create a function to model a single valve. Start with using a single pole pair in Eq. 5.1 and values given in Table 5.1. Use $F_s = 50$ kHz (or $dt = 1/F_s$), the sample rate after downsampling. Create functions to plot the time-domain and frequency-domain response.

   Python Tip: The Python SciPy signal toolbox has several useful functions.

   ```
   from scipy.signal import TransferFunction, lsim
   # Create the transfer function system
   system = TransferFunction(num, den)
   # Time array (from 0 to duration with step size of 1/Fs)
   ```

```
t = np.linspace(0, T_total, int(fs * T_total))
# Impulse response (time domain)
t_out, h_out, _ = lsim(system, U=np.ones_like(t), T=t)
```

Some other basic functions that may help you with these tasks are:

```
t, response = signal.step(system, T=t) # Step response
t, h = signal.impulse(system)  # impulse response
w, H = signal.freqresp(system) # Frequency response
tf2zpk(b, a)   # convert polynomial representation B(z)/A(z)
               # or B(s)/A(s) to poles and zeros
zpk2tf(z,p,k)  # convert zeros, poles and gain to B(z)/A(z)
               # or B(s)/A(s) representation
H = freqz(B,A) # compute frequency-domain response (z-domain)
H = freqs(B,A) # compute frequency-domain response (s-domain)
```

Read the manuals! And always first try out new functions on a toy example where you can predict the correct result.

Note that many functions cannot deal with poles in the RHS modeling anti-causal components (e.g., `filter`); instead, they will treat this as a causal but unstable component.

2. Add the individual responses to create a model of one of your measured signals. Start with the parameters given in Table 5.1 but feel free to adapt and change them with respect to your own recorded signals. Also insert the preprocessing Butterworth bandpass filter that you designed before.
   - Compare the time-domain plots to your earlier measurements. You can adjust your parameter settings to create a fit.
   - Similarly for the frequency-domain plots.
   - Comment on the quality of your fittings. Do you think a single-pole model for each valve is sufficiently accurate?
   - Make a array where the S1 and S2 sounds are repeated multiple times. To make it sound as realistic as possible, make sure the systole and diastole times are accurate.
3. Listen to your time-domain signal. Does it sound like the usual "lub-dub" stethoscope signal?
4. Test your model on your S1/S2 segmentation algorithm.
5. Optional extension: Python SciPy has an optimization toolbox `scipy.optimize`, e.g., `minimize`, `shgo`, and `least_squares`. You can try to use these to find optimal fitting values for your parameters. There may also exist specialized system identification toolboxes.

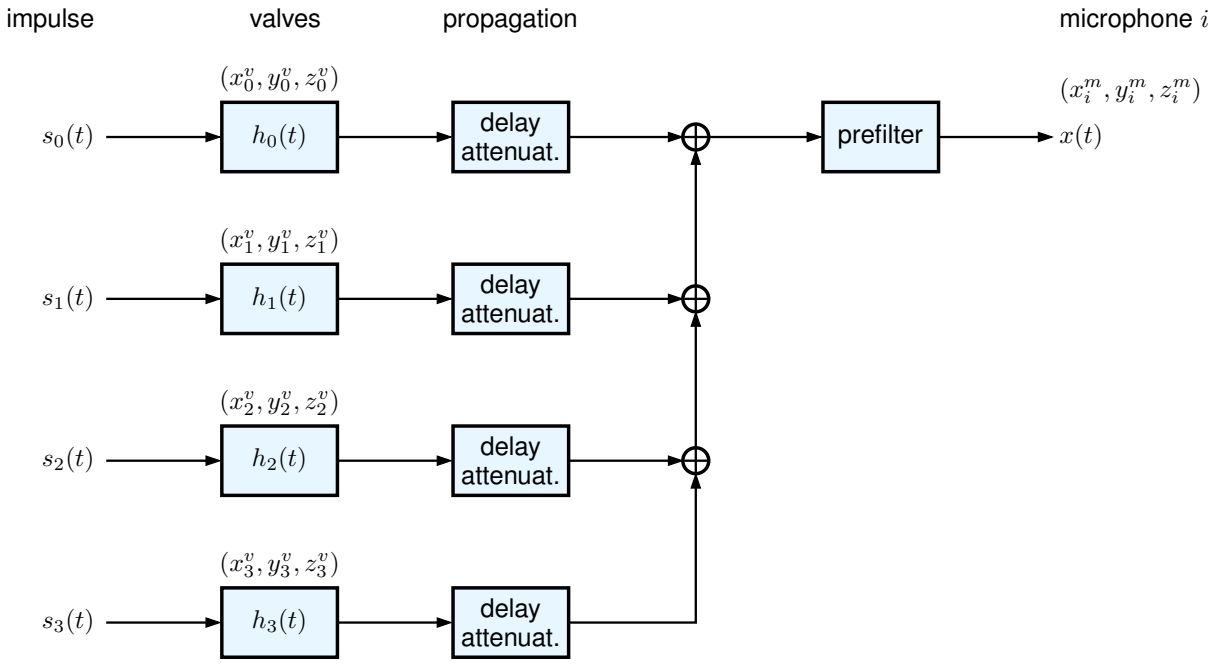


Figure 5.4. Model of the heart including a microphone measurement

5.3 MODELING MULTICHANNEL RECORDINGS

For System Integration, we will need a 3D propagation model to generate the measured response of the four valves at a microphone array. The signal of each valve propagates through the body and experiences a delay and an attenuation. Further, at each interface between two types of tissue, a part of the signal reflects and is scattered (viz. Labday 1 of the EE2S1 courselabs). In our simplified model, we will omit this effect.

A signal processing block diagram is shown in Fig. 5.4. As before, each valve is modeled as an LTI system, using one or more pole pairs. After delay and attenuation, the valve signals are added up at the microphone location. The resulting signal is measured and prefiltered, resulting in an observation $x_i(t)$, where i is the microphone index.

The propagation delay depends on the propagation distance between a valve and a microphone. Denote by $\mathbf{z}^v = [x^v, y^v, z^v]^T$ the coordinates of a valve, and by $\mathbf{z}^m = [x^m, y^m, z^m]^T$ the coordinates of a microphone, then

$$d^{vm} = \|\mathbf{z}^v - \mathbf{z}^m\| = \left((x^v - x^m)^2 + (y^v - y^m)^2 + (z^v - z^m)^2\right)^{1/2} .$$

To calculate the propagation delay, you will need the velocity of sound in the body. For the moment, let's assume $v = 60$ m/s. We will try to determine this value experimentally in Sec. **??**.

The attenuation depends on the propagation distance. Let's model this as a gain which is inversely

Table 5.2. Valve and microphone locations (example)

Valve	Location (cm)
0: M	(6.37, 10.65, 6.00)
1: T	(0.94, 9.57, 5.50)
2: A	(5.50, 11.00, 3.60)
3: P	(3.90, 11.50, 4.50)

Microphone	Location (cm)
mic 0	(2.5, 5.0, 0)
mic 1	(2.5, 10.0, 0)
mic 2	(2.5, 15.0, 0)
mic 3	(7.5, 5.0, 0)
mic 4	(7.5, 10.0, 0)
mic 5	(7.5, 15.0, 0)

proportional to the distance,

$$G^{vm} = \frac{1}{d^{vm}}\,.$$

Note that the delay and the attenuation depend on both the location of the valve and on the location of the microphone. To get started, you can use example values as in Table 5.2. We use a coordinate system where x is horizontal, y is height, and z is depth. The microphones are located at depth $z = 0$. In this example, there are 6 microphones in a 2×3 configuration; they have a spacing of 5 cm.

5.3.1 Assignments

1. Create a function *3d_model* in which you implement the 3D model, using the values as in Table 5.2. The model generates signals $x_m(t)$ for each microphone m.

   We suggest to stack these signals in a single matrix $\mathbf{X}$ consisting of 6 rows, where each row is the signal from one of the microphones, this matrix should be the output of the function.

2. Inspect the generated signals. Do they appear plausible?

3. Finish your model by implementing multiple heart beats! Try to randomize some of your model parameters to create slight changes in each beat.

4. Test your model on your S1/S2 segmentation algorithm.

5.4 INTEGRATION WITH OTHER MODULES

The output of this module will be used in other modules for segmentation and localization.

Simulated 6-channel recording: For segmentation in Module 1, you need to provide a 6-channel simulated recording that represents the signal as it would be heard by the microphone array on the chest of a patient. This can be then used for segmentation and localization using simulated data.

Simulated valve signals: These signals will be played through the two speakers of the phantom model. Note that you do not need to simulate delays, attenuation, or summation at the microphones, as this will

occur naturally when the sound travels through the phantom. You only need to provide the raw valve signals.

The phantom has only two speakers, so you can either:

- Play the Mitral (M) and Tricuspid (T) valve signals to create S1, or
- Play the Aortic (A) and Pulmonic (P) valve signals to create S2.

You cannot play all four valves simultaneously.

Our suggestion is to use one speaker for the M valve signal and one for the T valve signal. For better localization, it is recommended to increase the M valve frequency to 100 Hz and the T valve frequency to 300 Hz. While these values do not follow the real heart parameters, they better match the frequency range that the microphones and speakers can play and record effectively.

You can check Chapter 10 to learn and prepare yourself better for the actual recordings using the phantom.

5.5 EXTENSIONS

Once the main task is complete, you can consider the following extensions. Choose the one that interests you the most: the grade will depend on **quality**, not quantity. You are also welcome to propose your own extension ideas, but please discuss and confirm them with us before starting.

Improved heart sound modeling: Refine the heart sound models to better represent the acoustic properties of heart sounds and possible abnormalities.

Determining the speed of sound through the body: Different studies report varying values for the speed of sound through the chest. To investigate this, we have provided recordings (on Brightspace) for a simple experiment to estimate this speed. Keep in mind that the microphone may first record sound traveling through air (which is faster) before recording it through the body. Due to damping and reflections within the body, the sound may be less clear. Start with the provided recordings, but if results are unsatisfactory, consider conducting additional experiments to collect your own data.

During System Integration, we will attempt to localize the individual heart valves. We will do this by measuring differences in propagation delay. To translate this into distances, we will need to know the velocity of sound in the body.

Some known values are given in Table 5.3. As you can see, the shown values in the body are all above 1500 m/s. However, next to the heart we have the lungs, and with the stethoscope it appears we measure mostly via the lungs. The speed of sound in the lungs is not specified in the table.

In the literature, contradictory values are reported. The thorax has many components (solid tissues, airways, and lung parenchyma), with different velocities. The velocity also changes between inspiration and expiration. On average, researchers conclude that the sound propagates at an average speed of 23 to

Table 5.3. Velocity of sound [Binas 15A]

Material	Velocity [m/s]
Iron	5100
Water (293 K)	1483
Air (293 K)	343
Rubber	50
Blood	1580
Heart	>1570
Skin	>1730
Bone (porous)	1690–2410
Fat	1410–1490

70 m/s [4–6]. Can you verify this experimentally? To do so, you can design an experiment and record the data accordingly.

One approach to do so is to play an audio on one side of the chest and measure the response on a microphone on the other side, as it passes through the chest. After you have acquired the data, you can use deconvolution (your *ch3* algorithm) to find the channel impulse response $h(t)$, and determine the time delay (peak of the impulse response). Then calculate the velocity of sound. However, our previous experiments resulted in a speed similar to the speed of sound in air. This can show that the microphone will first record sound traveling through air (which is faster) before recording it through the body.

Murmur modeling and detection (multi-channel, simulation): In normal heart sounds, two valves (Mitral and Tricuspid) are heard during S1, and two other valves (Aortic and Pulmonic) during S2, with silent intervals during systole and diastole. In abnormal heart sounds, murmurs may occur during these normally silent phases. Common heart abnormalities include

- Mitral Valve Regurgitation
- Mitral Valve Stenosis
- Aortic Valve Regurgitation
- Aortic Valve Stenosis
- Ventricular Septal Defect (VSD)

Murmurs can be classified by timing, pitch, and location of the source. Although we do not provide our own recordings, there are data sets available with labeled single-channel audio recordings of normal heartbeats, murmurs, and noise.

Your task is to select an appropriate dataset, study the characteristics of murmurs, and model them based on signal features and source locations. You can then simulate your own data with murmurs, following

the same steps as in this module. The modeled signals can be used by the localization subgroup to develop murmur detection algorithms as discussed later in Section 9.4. It is important to study these abnormalities from a medical perspective before simulating them.

Alternatively, you can focus on murmur detection using single-channel data. This does not require modeling; instead, you can develop an approach based on murmur characteristics and S1/S2 properties. Be sure to consult with the instructor before starting coding to ensure that your chosen dataset and approach are suitable.

Bibliography

[1] M. Jabloun, P. Ravier, O. Buttelli, R. Ledee, R. Harba, and L.-D. Nguyen, "A generating model of realistic synthetic heart sounds for performance assessment of phonocardiogram processing algorithms," *Biomedical Signal Processing and Control*, vol. 8, no. 5, pp. 455–465, 2013.

[2] T. Tran, N. B. Jones, and J. C. Fothergill, "Heart sound simulator," *Medical and Biological Engineering and Computing*, pp. 357–359, May 1995.

[3] J. Xu, L. Durand, and P. Pibarot, "Nonlinear transient chirp signal modeling of the aortic and pulmonary components of the second heart sound," *IEEE Transactions on Biomedical Engineering*, vol. 47, no. 10, pp. 1328–1335, 2000.

[4] M. Kompis, H. Pasterkamp, and G. Wodicka, "Acoustic imaging of the human chest," *Chest*, vol. 120, pp. 1309–1321, Oct. 2001.

[5] H. Kajbaf and H. Ghassemian, "Acoustic imaging of heart using microphone arrays," in *Int. Conf. on Biomedical Engineering*, pp. 738–741, Springer, 2009.

[6] Y. Kawamura, Y. Yokota, and F. Nogata, "Propagation route estimation of heart sound through simultaneous multi-site recording on the chest wall," in *Annu. Int. Conf. IEEE Eng. Med. Biol. Soc.*, pp. 2875–2878, 2007.

Chapter 6

M3: BASIC DIRECTION FINDING

Contents

Using two ears, humans are quite capable to estimate the direction of a sound source. The main property that is used is the difference in arrival time at the ears. A less important property is slight changes in amplitudes. The shape of the ear lobes plays a role, as this allows to distinguish a source that is in the front from a source that is in the back. They make the angle-dependent arrival-time delays also frequency-dependent.

In this module, we will look at the basics of direction finding, which are valid not just for audio (including sonar and ultrasound imaging), but also for RF signals, as e.g. used in radar, or radio astronomy interferometry. For the moment, we will look at a single source in a 2D world (plane), such that we have to estimate just a single parameter: its azimuth angle. During System Integration, you will attempt to apply this to the localization of heart valves. Not at all easy, because in comparison to 'classical' theory, this application scenario is near-field, wide-band, and non-stationary.

Generally, the theory that we cover is considered advanced; it is known as phased array processing and it is best studied after a course on random processes (EE3S1). At TU Delft we teach this in the master as EE4715 Array Processing. Since the array signals are stacked in vectors, also linear algebra plays an important role.

Learning objectives The following is learned and practiced in this module:

- Basic theory of phased array processing
- Measuring and estimation of the direction of a single acoustic source using a linear microphone array

Preparation

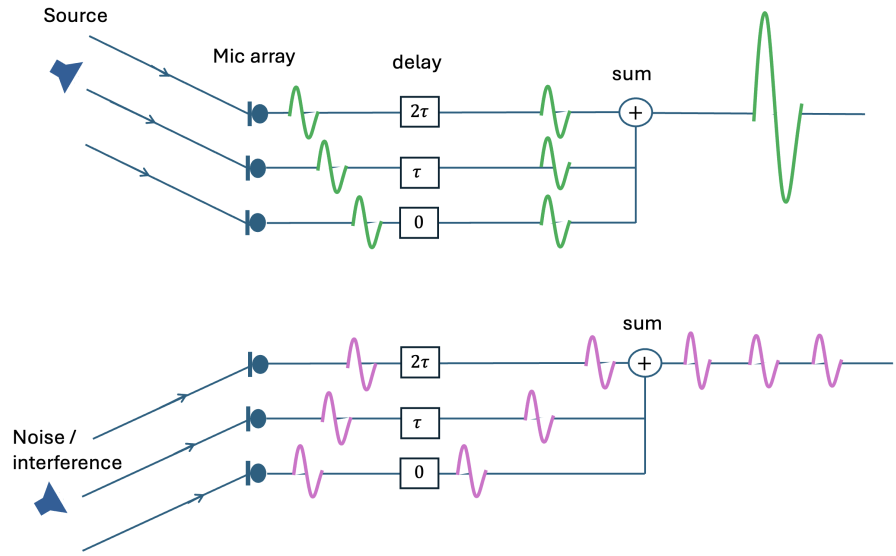


Figure 6.1. A source and a noise/interference signal received at three microphones, followed by a delay-and-sum beamformer.

- Read this module; it has quite a lot of new theory
- Brush up on linear algebra!

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- Data recorder
- Linear microphone array with 6 microphones

6.1 ARRAY PROCESSING BASICS

Array signal processing is a powerful technique that involves the analysis of multiple sensors, such as microphones or antennas, arranged in a specific configuration to capture and process signals. In the context of audio, this is often done with multiple microphones working together to analyze sound coming from different directions. This provides several key advantages over using just a single microphone:
Direction finding and source localization: With multiple microphones, we can determine the direction and/or location of the sound source. This is because the time differences in the arrival of the sound at

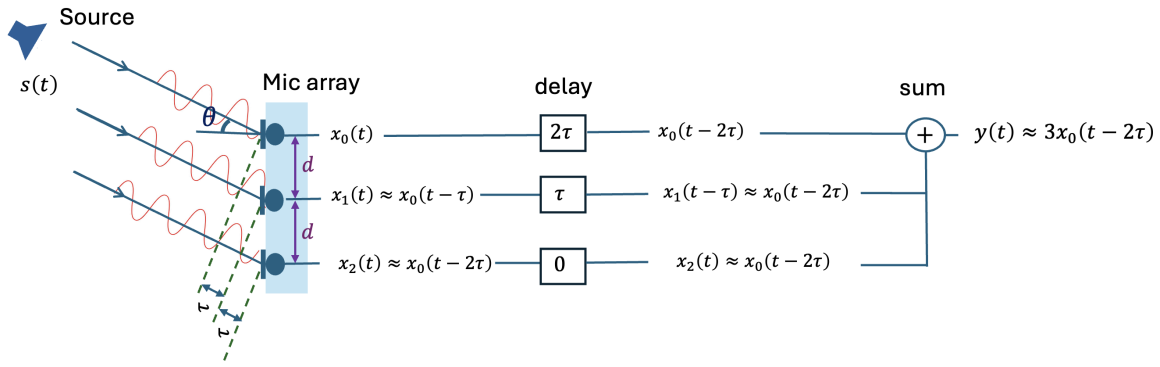


Figure 6.2. A signal received at three microphones, followed by a delay-and-sum combiner.

each microphone provide information about where the sound is coming from. This is useful for applications like smart speakers or voice-controlled systems, which need to "know" where the user is speaking from.

Signal enhancement (Beamforming) and noise reduction: With multiple microphones, we can determine the direction from which a sound is coming. By processing the signals from each microphone, it's possible to focus on sounds from a specific direction (such as someone's voice) and reduce noise or interference from other directions (also called spatial filtering).

6.2 BEAMFORMING

Beamforming is an array signal processing technique used in systems with multiple sensors (such as microphones or antennas) to focus on signals arriving from a specific direction. It enhances the reception of signals from a desired direction while minimizing interference from other directions.

6.2.1 Delay-and-sum beamforming

In delay-and-sum beamforming, multiple sensors receive the same signal, but with slight time differences. This time difference occurs because the signal reaches each sensor at a different moment, depending on its position relative to the source. An example with three microphones is shown in Fig. 6.1.

Delay: The first step in delay and sum beam forming is to apply time delays to the signals received by the sensors. These delays are calculated so that the signals from the desired direction become **aligned** in time.

Sum: Once the signals are aligned, they are summed together. This reinforces the signal from the desired direction because all the aligned signals add up coherently (in phase), making it stronger.

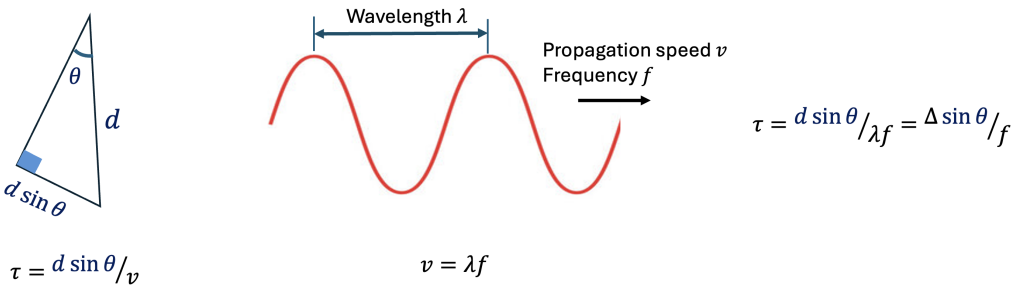


Figure 6.3. Finding the delay between two microphones based on θ and $\Delta = d/\lambda$.

By delaying and summing the signals, the beamformer amplifies the signal from the desired direction, while signals from other directions remain unaligned and, when summed, tend to cancel each other out or become less significant. This selective enhancement of the signal from a particular direction improves the clarity of the signal and reduces interference from unwanted sources. Fig. 6.1 shows an example of a source and a noise/interference signal received by the beamformer and its outputs.

In the example shown in Fig. 6.2 we have a linear array of three microphones positioned along a straight line, spaced apart by equal distances d. In this model, the source is assumed to be in the *far field*, far enough from the microphone array that the wavefronts arriving at the array can be considered parallel. This also means that there is a single angle-of-arrival θ valid for all microphones. The far-field assumption also means that there are no amplitude differences among the microphone signals (at least, if their gains have been properly calibrated).

The unknown source signal $s(t)$ is received over all microphones. Since we don't know the source, let us call the signal received at the first microphone $x_0(t) = s(t)$; this will be our reference signal. The signal at the second microphone arrived a bit later: $x_1(t) = s(t-\tau)$, and at the third $x_2(t) = s(t-2\tau)$.

As shown in Fig. 6.3, if we know τ, the distance between the microphones d, and the speed of sound v, then we can estimate the angle-of-arrival θ via the relation $\tau = (d/v)\sin(\theta)$. Alternatively the delay equation may also be written in terms of $\Delta = d/\lambda$, the microphone spacing expressed in wavelengths then $\tau = (\Delta/f)\sin(\theta)$. Further, we can perform delay and sum beamforming and thus remove noise. We denote the output of the beamformer by $y(t)$.

During the Labdays in the first week, you have already seen how we can estimate delay τ: via deconvolution. We first find $h(t)$, and then look for the location of the first peak to find τ. In practice, it is hard to work with delays. In analog time domain, there are no electronic components that generate a delay, except for (very long) cables.[1]

A second approach to find the direction of the source (or their equivalent delays) is to look at the power of the output of the delay and sum beamformer $y(t)$ at all possible directions θ. The idea is that the beam-

[1] E.g., in the Westerbork telescope, the incoming signals were first aligned by kilometers of cable in the basement.

former output power is maximized if θ matches the true source direction due to aligning and summing the received signals.

With this introduction, we now aim to derive the equation for the output power of the beamformer P_y based on θ. By maximizing this equation we can therefore find the true source direction θ_0. We start with deriving the data (input) model, next the beamformer output model and finally the output power model.

6.2.2 Data model

In array processing, signals are stacked in vectors. E.g., for three microphones, we can write the data model as

$$\mathbf{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} s_0(t) \\ s_0(t-\tau) \\ s_0(t-2\tau) \end{bmatrix}.$$

As noted earlier it is hard to work with delays in time, however if the source is of form $s(t) = e^{j\Omega_0 t}$ (complex sinusoid), the situation simplifies significantly as

$$s(t-\tau) = e^{j\Omega_0(t-\tau)} = e^{-j\Omega_0\tau} s(t).$$

where the delay appears as a phase shift $e^{-j\Omega_0\tau}$ in the time domain and the signal at each microphone can simply be written as the original signal $s(t)$ multiplied by a phase factor that depends on the delay. Therefore we can perform *phased array* processing and work with the phase factors $\exp(-j\Omega_0\tau)$ to overcome the issues of working with the delays in time directly.

Therefore the data model can be written compactly as

$$\mathbf{x}(t) = \begin{bmatrix} x_0(t) \\ x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} s_0(t) \\ s_0(t-\tau) \\ s_0(t-2\tau) \end{bmatrix} = \begin{bmatrix} 1 \\ e^{-j\Omega_0\tau} \\ e^{-j\Omega_0 2\tau} \end{bmatrix} s(t) = \mathbf{a} s(t).$$

The vector $\mathbf{a}$ contains the phase factors (the effects of the propagation channel. It is called the *array response or steering vector*. Since it depends on the angle-of-arrival, we often write $\mathbf{a}(\theta)$.

Written in this form, we can immediately extend to more than three microphones! For M microphones, equally spaced by d, we stack the signals $x_i(t)$ into a vector $\mathbf{x}(t)$ with M entries, and we find for the array response vector

$$\text{array response vector: } \mathbf{a}(\theta) = \begin{bmatrix} 1 \\ e^{-j(d/v)\sin(\theta)\Omega_0} \\ e^{-j2d/v\sin(\theta)\Omega_0} \\ \vdots \\ e^{-j(M-1)(d/v)\sin(\theta)\Omega_0} \end{bmatrix}. \tag{6.1}$$

In this notation, we made the dependency of $\mathbf{a}(\theta)$ on θ explicit.

It is not hard to generalize this to other, non-equally spaced array configurations, and non-uniform arrays often have better performance. The reason we often assume a Uniform Linear Array (ULA) is that it is so easy to specify the arrangement.

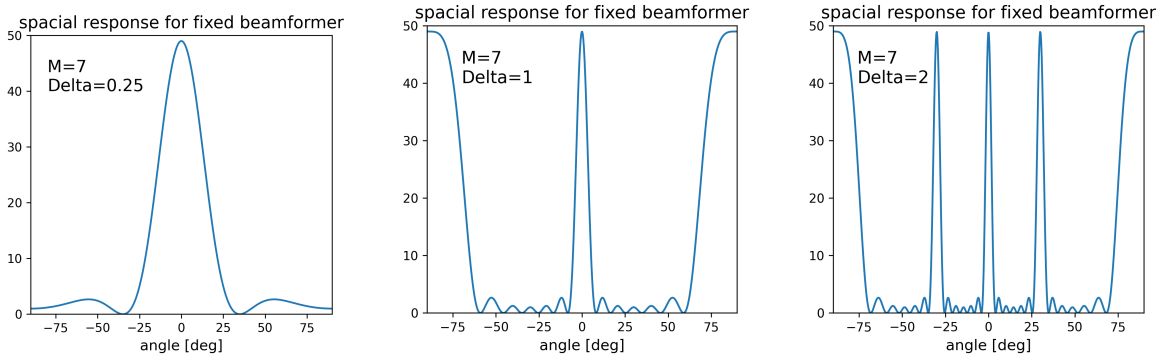


Figure 6.4. Grating lobes. $M = 7, \theta_0 = 0°$. If $\Delta = d/\lambda$ is smaller than $1/2$, then resolution is reduced. If $\Delta = d/\lambda$ is larger than $1/2$, then spatial aliasing occurs.

6.2.3 Assignments

Create a Python function `a_lin(theta, M, d, v, f0)` which generates $\mathbf{a}(\theta)$ for M uniformly spaced microphones at distances d, with v the speed of sound and f_0 center frequency of the wave in Hertz ($\Omega_0 = 2\pi f_0$). Test your function with different values to make sure it works properly.

6.2.4 Beamformer output model

The delay-and-sum beamformer needs to align the phases and then sum up. In phased array processing we can implement this by pre-multiplying $x_1(t)$ with $e^{+j\Omega_0\tau}$ and $x_2(t)$ with $e^{+j\Omega_0 2\tau}$ To cancel out the phase shift. Note that in time domain it will be like advancing the signals and not delaying them but will have the same effect.

$$y(t) = x_0(t) + e^{+j\Omega_0\tau}x_1(t) + e^{+j\Omega_0 2\tau}x_2(t) = \begin{bmatrix} 1 & e^{+j\Omega_0\tau} & e^{+j\Omega_0 2\tau} \end{bmatrix} \begin{bmatrix} x_0(t) \\ x_1(t) \\ x_2(t) \end{bmatrix} = \mathbf{a}^H\mathbf{x}(t)\,.$$

Note that superscript ^{H} denotes a complex conjugate transpose. The conjugation is needed to compensate for the phase. Therefore, the general form for M microphones will be

$$\begin{aligned} \text{data model:} \quad & x(t) = \mathbf{a}(\theta)\mathbf{s}(t)\,. \\ \text{beamformer output:} \quad & y(t) = \mathbf{a}(\theta)^H\mathbf{x}(t)\,. \end{aligned}$$

6.2.5 Output power

Let us now consider the situation with a single source $s_0(t)$, coming from direction θ_0:

$$\mathbf{x}(t) = \mathbf{a}(\theta_0)s_0(t)$$

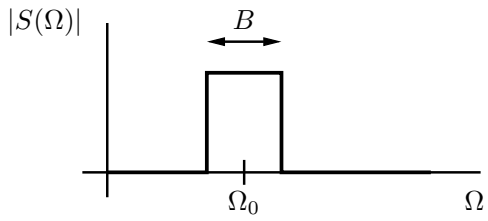


Figure 6.5. A narrowband signal.

and the beamformer output at the direction θ is

$$y(t) = \mathbf{a}(\theta)^H \mathbf{x}(t) = \mathbf{a}(\theta)^H \mathbf{a}(\theta_0) s_0(t)\,.$$

As noted earlier, to find the source direction we can look at the power of the output of the beamformer $y(t)$ at all possible directions θ. While scanning θ, the idea is that the "response" has a peak if θ matches the true source direction, $\theta = \theta_0$. Here, "response" is defined as the *output power* P_y of $y(t)$.

There are several ways to define P_y. In a stochastic setting (where we consider random signals), the definition is $P_y = \mathrm{E}[|y(t)|^2]$. You will study random signals later in EE3S1. Under suitable conditions (stationarity, ergodicity), it can be replaced by the average for a sampled signal $y[n]$ as,

$$P_y = \lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} |y[n]|^2$$

We can define the power of the input signal $s_0(t)$ in the same way. Let's for now assume $s_0(t)$ has unit power. Then,

$$P_y(\theta) = |\mathbf{a}^H(\theta)\mathbf{a}(\theta_0)|^2 \cdot 1\,.$$

This is called the **spatial spectrum**, and can be plotted as a function of θ, see Fig. 6.4 for different examples. Here the peak shows the direction where the source is coming from.

6.2.6 Assignments

1. Use your function `a_lin(th, M, d, v, f0)` to recreate the plots shown in 6.4. Note that you can use any default value that for v and f_0, for example $v = 340m/s$ (the speed of sound in air) and $f_0 = 500Hz$. While d can be calculate using the given Δ as $d = \lambda * \Delta = (v/f) * \Delta$.

   Tip: we have complex matrices, and usually you will need the complex conjugate transpose ("Hermitian"), rather than the normal transpose. In Python, write `a.conj().T`

2. Change the source direction and investigate the plots.

6.2.7 Narrowband condition

So far, for phased array processing, we assumed that the input signal is a complex sinosuid. However, we would like to use more general (analytic) signals than that. We can show that (see A.1 for proof) for a sum of sinusoids $s(t) = \int S(\Omega)e^{j\Omega t}\mathrm{d}\Omega$ A.1 each with their own amplitude $S(\Omega)$ the shift in time can be implemented as phase factor if the support of $S(\Omega)$ is the interval $[\Omega_0 - B/2, \Omega_0 + B/2]$, where

$$B\,[\mathrm{Hz}] \ll \frac{v}{D}$$

with $D = (M-1)d$ be the size of the array (the maximal baseline). This is called the *narrowband condition*.

If the narrowband condition is not satisfied (which quickly happens for audio signals), then we need to resort to *wideband array processing*. The usual solution is to apply a bandpass filter, such that the output of the bandpass satisfies the narrowband condition. We then vary the center frequency of the bandpass over a range of values. A convenient way to do this is to apply the Short-term Fourier Transform (STFT), which is your spectrogram, but without taking the absolute values. We will investigate this in more details later in Section 6.6.

6.2.8 Microphone spacing

Each microphone samples the input signal in time. However, the microphone array samples the input signal in space, and just as with sampling in time, the issue of aliasing comes up. Thus, to avoid spatial aliasing, we have to sample more densely in space than half the wavelength (see A.2 for proof). Alternatively, if we use $\Delta = d/\lambda$, the microphone spacing expressed in wavelengths, we need $\Delta < 1/2$.

A lot of array processing literature makes a standard assumption of half wavelength spacing; however, this is certainly not always practical. During wideband processing, if we have a filterbank with several center frequencies, then Ω_0 varies, and therefore also λ varies.

6.2.9 Assignments

1. Later in this chapter we will experiment with a linear array of 6 microphones, spaced by 10 cm. What is the condition on B [Hz] for narrowband condition to hold?

2. For our linear array (6 microphones), what is the minimal spacing between the microphones such that spatial aliasing is avoided? Take $F_0 = 1000$ Hz.

3. For our heart sound localization application, we have a 2×3 array with spacing of 5 cm. What is the condition on B [Hz]? you can assume that v_{body} is 60m/s.

4. Looking back at the plots in 6.4 explain what happens if you take Δ smaller or larger. Try to think of the physical reason why this happens.

6.3 OPTIMAL BEAMFORMERS

We are now ready for our goal in this module: direction finding. What we saw so far was a specific implementation of beamforming: delay and sum. In this section, we first present a general formulation of beamforming. And then introduce two types of beamforming, matched beamforming and MVDR.

6.3.1 Beamformer: general representation

A more general formulation of beamforming is

$$y(t) = \mathbf{w}^H \mathbf{x}(t) = \bar{w_0} x_0(t) + \bar{w_1} x_1(t) + \cdots + \bar{w_M} x_M(t)$$

where the beamformer $\mathbf{w}$ could be any vector which combines the microphone signals (weighted summation) and $\bar{w_0}$ denotes the complex conjugate of w_0.

Note that while $\mathbf{w}$ (the beamforming weights) could be "any" vector in the mathematical sense, in practice, it is chosen with a specific objective in mind to achieve our desired outcome depending on the application. Here are some common objectives for beamforming including: maximizing signal from a desired direction (A delay-and-sum beamformer we saw earlier in an example), minimizing interference from other direction (the MVDR beamformer is an example here that we will investigate later in this chapter), maximizing Signal-to-Noise Ratio (SNR), minimizing output power, achieving spatial filtering, etc.

In a computer, we will always work with sampled signals. Therefore, in this section, we extend our models for the sampled data. For the output model, we define a (row) vector of output samples, $\mathbf{y} = [y[0], \cdots, y[N-1]]$, then we can write $\mathbf{y}$ in terms of the data matrix $\mathbf{X}$ as

$$\mathbf{y} = \mathbf{w}^H \mathbf{X}.$$

$$\begin{bmatrix} y[0] & y[1] & \cdots & y[N-1] \end{bmatrix} = \begin{bmatrix} \bar{w}[0] & \bar{w}[1] & \cdots & \bar{w}[M-1] \end{bmatrix} \begin{bmatrix} x_0[0] & x_0[1] & \cdots & x_0[N-1] \\ x_1[0] & x_1[1] & \cdots & x_1[N-1] \\ \vdots & \vdots & \vdots & \vdots \\ x_{M-1}[0] & x_{M-1}[1] & \cdots & x_{M-1}[N-1] \end{bmatrix}$$

In this expression, $\mathbf{w}^H$ combines the rows of $\mathbf{X}$ into a single output signal. Matrix $\mathbf{X}$ of size $M \times N$ includes all the samples $(n = 0, .., N-1)$ of the all microphones' recording $(m = 0, .., M-1)$.

In terms of the output power P_y, we find, using expectations (The details of this will come in EE3S1.),

$$P_y = \frac{1}{N} \mathbf{y}\mathbf{y}^H = \frac{1}{N} \mathbf{w}^H \mathbf{X}\mathbf{X}^H \mathbf{w} = \mathbf{w}^H \mathbf{R}_x \mathbf{w}$$

where $\mathbf{R}_x = \frac{1}{N}\mathbf{X}\mathbf{X}^H$ is the covariance matrix.

6.3.2 Matched beamformer

For the matched beamformer, which we saw earlier, we take $\mathbf{w} = \mathbf{a}(\theta)$. The spatial response, as plotted earlier in Fig. 6.4, is the function

$$P_y(\theta) = \mathbf{a}^H(\theta)\mathbf{R}_x\mathbf{a}(\theta)\,.$$

We look for the peaks of this function (maximum output power), and these point at the source directions. This can be written as an optimization problem

$$\hat{\theta} = \arg\max_{\theta}\ \mathbf{a}^H(\theta)\mathbf{R}_x\mathbf{a}(\theta)\,.$$

the estimated source direction $\hat{\theta}$ ($\hat{}$ denotes the estimated value for θ) is the angle that maximizes $\mathbf{a}^H(\theta)\mathbf{R}_x\mathbf{a}(\theta)$.

Actually, this is only optimal if there is just a single source! The optimization problem tries to align $\mathbf{a}(\theta)$ to the dominant eigenvector of $\mathbf{R}_x$, which is equal to a source direction vector only for a single source (or approximately, for widely spaced sources).

6.3.3 MVDR

If we expect more than a single source, then an idea is to treat the additional sources as *interferers*. The beamformer can act as a spatial filter which suppresses (nulls) the interferers.

The idea is implemented as follows. Suppose we want to see if there is a signal coming from direction $\mathbf{a}(\theta)$. We define the beamformer $\mathbf{w}$ to be such that the response to a signal from this direction is equal to a constant, 1:

$$\mathbf{w}^H\mathbf{a}(\theta) = 1\,.$$

This equation ensures that the signal coming from direction θ is passed through the beamformer with a gain of 1, meaning that there is no distortion and the signal from this direction is preserved perfectly after the beamforming process.

This constraint does not completely specify $\mathbf{w}$. We use the remaining degrees of freedom in defining $\mathbf{w}$ such that it will minimize the output power, which means the power due to the interfering signals. Note that, this will not affect the signal of interest (from the current look direction θ) due to the constraint $\mathbf{w}^H\mathbf{a}(\theta) = 1$. Therefore the optimization problem is

$$\mathbf{w}_{\text{opt}}(\theta) = \arg\min_{\mathbf{w}} \mathbf{w}^H\mathbf{R}_x\mathbf{w} \qquad \text{such that} \quad \mathbf{w}^H\mathbf{a}(\theta) = 1\,.$$

This can be interpreted as, we are looking for an optimal value for the beamformer $\mathbf{w}_{\text{opt}}(\theta)$ that minimizes the overall output power (due to interference) while preserving the the signal from the desired direction θ. It can be shown that the solution of this optimization problem is:

$$\mathbf{w}_{\text{opt}} = \frac{\mathbf{R}_x^{-1}\mathbf{a}(\theta)}{\mathbf{a}^H(\theta)\mathbf{R}_x^{-1}\mathbf{a}(\theta)}$$

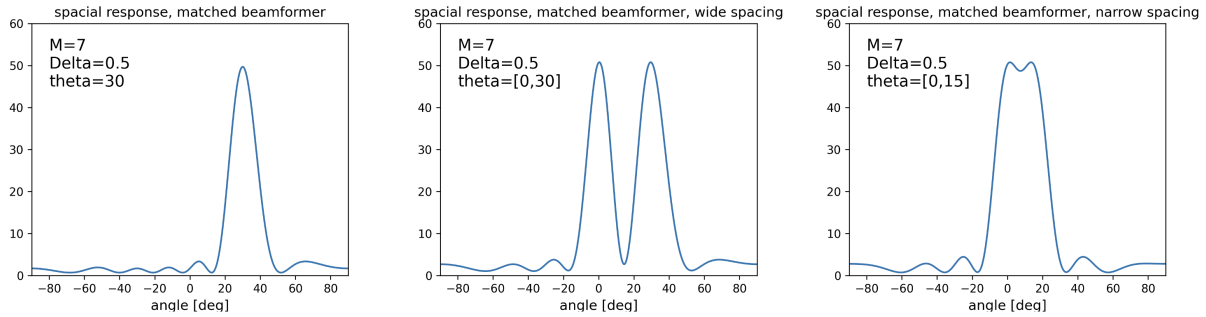


Figure 6.6. Direction finding. (a) one source; (b) two sufficiently spaced sources; (c) two closely spaced sources.

This is the optimal weight vector for the MVDR beamformer.

Here, the numerator $\mathbf{R}_x^{-1}\mathbf{a}(\theta)$ computes the directionally focused beam pattern, weighted inversely by the covariance of the received signals (which includes the interference). Meanwhile, the denominator $\mathbf{a}^H(\theta)\mathbf{R}_x^{-1}\mathbf{a}(\theta)$ normalizes the beamformer to ensure that the gain in the direction θ is 1, preserving the signal from the desired direction without distortion. The corresponding spatial response is

$$P_y(\theta) = \mathbf{w}^H\mathbf{R}_x\mathbf{w} = \frac{1}{\mathbf{a}^H(\theta)\mathbf{R}_x^{-1}\mathbf{a}(\theta)}\,.$$

6.4 TWO SOURCES

As previously mentioned, the advantage of using MVDR over a matched beamformer becomes apparent in scenarios with multiple sources. To compare these two beamformers, we will consider a situation with two sources,

$$\mathbf{x}(t) = \mathbf{a}(\theta_0)s_0(t) + \mathbf{a}(\theta_1)s_1(t)$$

As before we develop the model for sampled signals. After sampling, the data model for a single source is

$$\mathbf{x}[n] = \mathbf{a}\,s[n]\,, \qquad n = 0,\cdots,N-1\,.$$

Next, we would always stack multiple samples in a vector, and multiple vectors in a matrix:

$$\mathbf{X} = [\mathbf{x}[0],\cdots,\mathbf{x}[N-1]]\ :\ M\times N\,, \qquad \mathbf{s} = [s[0],\cdots,s[N-1]]\ :\ 1\times N\,.$$

Note that we usually prefer vectors to be column vectors (just to avoid confusion), but $\mathbf{s}$ is a row vector. The data model is

$$\mathbf{X} = \mathbf{a}\,\mathbf{s}\,.$$

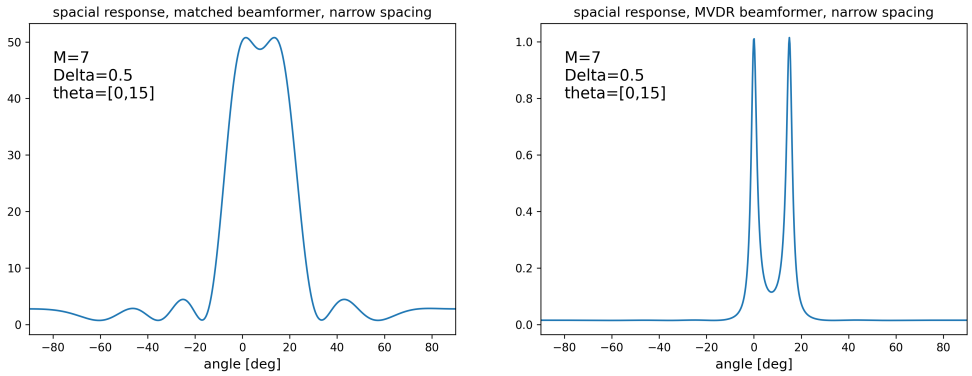


Figure 6.7. Direction finding. Spatial spectrum for (a) matched beamformer; (b) MVDR, SNR = 10 db.

Here, $\mathbf{X}$ is an $M \times N$ matrix, while $\mathbf{a}$ is a column vector and $\mathbf{s}$ is a row vector; the product $\mathbf{as}$ is an "outer product" and results in a matrix. Since all columns of $\mathbf{X}$ are simply scaled versions of $\mathbf{a}$, the matrix $\mathbf{X}$ is rank 1.

Working in the same way, for two sources, the data model is

$$\mathbf{X} = \mathbf{a}_0 \mathbf{s}_0 \; + \; \mathbf{a}_1 \mathbf{s}_1$$

We can write this compactly as

$$\mathbf{X} = \mathbf{A}\mathbf{S}, \qquad \mathbf{A} = [\mathbf{a}_0 \;\; \mathbf{a}_1] \; : \; M \times 2, \qquad \mathbf{S} = \begin{bmatrix} \mathbf{s}_0 \\ \mathbf{s}_1 \end{bmatrix} \; : \; 2 \times N$$

(Verify this!) Since all columns of $\mathbf{X}$ are a linear combination of $\mathbf{a}_0$ and $\mathbf{a}_1$, we know that $\mathbf{X}$ is rank 2. Hence, we conclude that the rank of $\mathbf{X}$ tells us the number of sources!

Fig. 6.7 shows an example which compares the matched beamformer to the MVDR, for two closely spaced sources. It is seen that, indeed, the MVDR has higher resolution. In the simulation, noise was added or else the peaks would go to infinity!

6.4.1 Assignments

1. Implement functions which generate the spatial spectra of the matched beamformer and of the MVDR:

   ```
   P_mbf = matchedbeamformer(Rx, th_range, M, d, v, f0)
   P_mvdr = mvdr(Rx, th_range, M, d, v, f0)
   ```

   Here, `th_range` is a vector of angles on which the spectra are evaluated. To generate the data covariance matrix $\mathbf{R}_x$ in the presence of noise, you can use

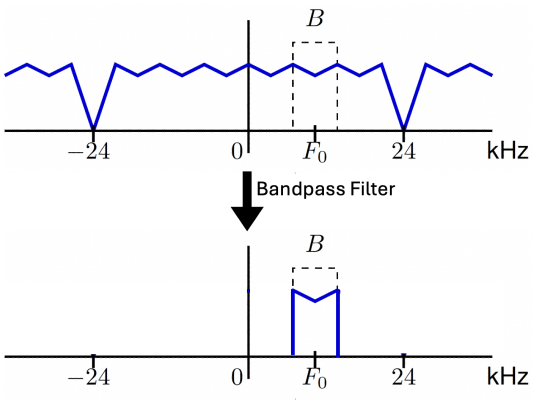


Figure 6.8. Bandpass filtering for creating narrowband signals.

```
sigma_n = 10**(-SNR/20);      # std of the noise (SNR in dB)
A = a_lin(theta0, M, d, v, f0); # source direction vectors
A_H = A.conj().T
R = A @ A_H                   # assume equal powered sources
Rn = np.eye(M,M)*sigma_n**2; # noise covariance
Rx = R  + Rn                  # received data covariance matrix
```

Here, `theta0` is a vector with true source angles where the number of sources is equal to `Q = len(theta0)`. Make sure A the output of your `a_lin` is a 2 dimensional Numpy array of shape $M \times Q$

2. Reproduce the plots in Fig. 6.7.

3. If θ_0 is one of the source angles, find the corresponding MVDR beamformer $\mathbf{w}_{\text{opt}} = \frac{\mathbf{R}_x^{-1}\mathbf{a}(\theta)}{\mathbf{a}^H(\theta)\mathbf{R}_x^{-1}\mathbf{a}(\theta)}$, and make a plot of the spatial response of this beamforming vector. By this, show that the response to the second source is zero.

6.5 FROM WIDEBAND TO NARROWBAND

As discussed previously, all of our developed DoA (Direction of Arrival) algorithms are designed for narrowband signals. However, the recording we are dealing with here is wideband. To handle this, we can process the signal in separate frequency bands.

One simple approach is to select a central frequency f_0 and a bandwidth B that satisfy the narrowband assumption, then filter the signal within that band. Figure 6.8 illustrates this procedure. While straightforward, this method is not very efficient.

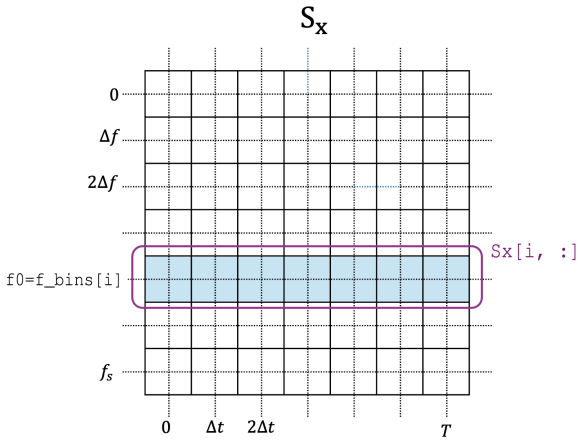


Figure 6.9. Sx matrix and the bandpass signal in frequency domain.

A more efficient approach is to use the Short-Time Fourier Transform (STFT), which provides a frequency-by-time representation of the signal through the STFT magnitude *Sx*. An example Python implementation is:

```
from scipy.signal import ShortTimeFFT
from scipy.signal.windows import gaussian

win = ('gaussian', 1e-2 * fs)  # Gaussian window with 0.01 s standard deviation
SFT = ShortTimeFFT.from_window(win, fs, nperseg, noverlap=0,
                               scale_to='magnitude', phase_shift=None)
Sx = SFT.stft(signal)
f_bins = SFT.f
```

By computing the STFT of a single recording, we obtain a matrix `Sx` of dimensions (number of frequency bins) $\times$ (number of time slices), where each row corresponds to a narrowband signal of frequency resolution Δf centered at `f0=f_bins[i]` over time. Therefore, to get the narrowband signal, we can simply use `Sx[i, :]`. Figure 6.9 illustrates this. There is no need to perform an inverse STFT, as the steering vector is independent of time, allowing us to work directly with the STFT magnitude.

For our purposes, the important parameters are the bandwidth and the central frequency. While there is no parameter to set the bandwidth directly in the code above, it is determined by `nperseg`, which specifies the number of samples per FFT segment (i.e., the window length in samples). The frequency

resolution (i.e., effective bandwidth of each bin) is given by:

$$\Delta f = \frac{f_s}{\text{nperseg}}$$

This implies:

- Larger `nperseg` $\rightarrow$ finer frequency bins (better frequency resolution).
- Smaller `nperseg` $\rightarrow$ coarser frequency bins (poorer frequency resolution).

The central frequency of each bin is then determined by its index k:

$$f_k = k\Delta f, \quad k = 0, 1, \ldots, \texttt{nperseg} - 1.$$

There is no need to calculate these manually, as they are provided in the output `f_bins`. By inspecting `f_bins`, you can select the index `i` corresponding to your desired central frequency `f0=f_bins[i]`.

For more efficient matrix calculations, we can compute the STFT for all microphone recordings, producing a matrix `Sx_all` of dimensions (number of microphones) $\times$ (number of frequency bins) $\times$ (number of time slices).

We can then compute the narrowband covariance matrix $\mathbf{R}_x$ directly in the frequency domain. For a selected central frequency `f0=f_bins[i]`, the narrowband data matrix is `X = Sx_all[:, i, :]`, with dimensions (number of microphones) $\times$ (number of time slices). Its covariance is then obtained as `Rx = np.cov(X)`. While this may seem complex in text form, it becomes much clearer when you write out the equations and work through them step by step. Try doing this yourself before coding, as it helps deepen your understanding.

Finally, $\mathbf{R}_x$ can be used in MVDR beamforming as `mvdr(Rx, th_range, M, d, v, f0)`. In practice, for better results, $\mathbf{R}_x$ can be computed for multiple bandwidths and central frequencies by adjusting `nperseg` and selecting different `i` indices. Comparing the resulting spatial responses allows one to determine which values provide the best DoA estimation. Additionally, averaging spatial responses across multiple selected frequencies can yield more robust results. This procedure will be demonstrated in the next experiment.

6.6 EXPERIMENTS

Now that your simulations are operational, you are ready to test your beamforming algorithms on experimental data. We will be using the *uniform linear microphone array*, discussed in Sec. 3.3.

6.6.1 Measurement

It has array spacing$d = 10$ cm. We will first measure with one loudspeaker, and then extend to two sources.

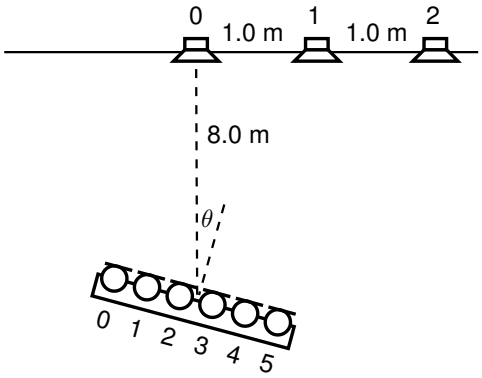


Figure 6.10. Experimental setup.

The setup is shown in Fig. 6.10. Source 0 is fixed on the wall. The array is placed at a distance of 8 m away from the wall, on the center line. For practical reasons, rather than rotating the source around the array, we will rotate the array, just like a radar dish.

We first make recordings using a single source, but place this source on 3 different positions.

1. Generate an audio sound file with white noise (`randn`, use real-valued noise this time). The sample rate should be $F_s = 48$ kHz, it should have a duration of about 10 s. Check that you can play the sound file on your laptop. Create 3 different random versions, for source position 0, 1 and 2.

2. Create the setup for a single source. Make audio recordings for a variety of angles θ, from 0° till 60° in steps of 20°. Use filenames which remind you of the setting, and make notes.

3. Now, add another source (e.g. on position 2 in Fig. 6.10, 1 m away from the original speaker) to play two **different** white noise samples at the same time. You can use this setup to test your algorithm on multi-source separation, much like the heart!

This completes the experiment. Next, you are going to apply your algorithms, and check if you can recover the direction-of-arrival!

6.6.2 Beamforming

Following the description and steps in Sec. 6.5 complete the following items.

1. Implement a function `f_bins, Rx_all= narrowband_Rx(.)` following the steps explained above to generate $\mathbf{R}_x$ for different frequency bands.

2. Apply your beamforming algorithms (matched beamforming and MVDR) and see how accurately you can recover the direction-of-arrival of the source. Compare different values for the bandwidth and center frequency for best DoA estimation. Document all your results and values. Note that in

your report, you have to discuss and present the values of your design in terms of central frequency and bandwidth. Avoid using parameters you use in your code like `i` or `nperseg`.

Chapter 7

M4: ADVANCED DIRECTION FINDING

Contents

In Module 3, you have been introduced to some basic direction finding algorithms, and seen that they are adequate to separate two sources on the linear microphone array. The methods are based on estimating phase differences. But if you now consider in more detail our intended application (heart sound separation), you will see that the phase differences will be much smaller! We will have a need for *super-resolution* algorithms. In this chapter, we will look at MUSIC.

Learning objectives The following is learned and practiced in this module:

- More advanced array processing: MUSIC

Preparation

- Read this module

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- Your measurement data from Module 3

7.1 SINGULAR VALUE DECOMPOSITION

Before exploring higher resolution direction finding algorithms, it's essential to delve deeper into the data matrices and review some foundational on singular value decomposition.

7.1.1 Data matrix

As you saw earlier in section 6.4 the data matrix for two sources can be written compactly as

$$\mathbf{X} = \mathbf{A}\,\mathbf{S}:\ M \times N, \qquad \mathbf{A} = [\mathbf{a}_0\ \ \mathbf{a}_1 \cdots;\ \mathbf{a}_Q]\ :\ M \times Q, \qquad \mathbf{S} = \begin{bmatrix} \mathbf{s}_0 \\ \mathbf{s}_1 \\ \vdots \\ \mathbf{s}_Q \end{bmatrix}\ :\ Q \times N \tag{7.1}$$

where Q is the number of sources. An noted earlier, all columns of $\mathbf{X}$ are a linear combination of $\mathbf{a}_0$ to $\mathbf{a}_Q$. Therfore, we know that $\mathbf{X}$ is rank Q , since it represents the contributions from Q independent sources. Let's look more into the rank of the matrix.

7.1.2 Singular value decomposition (SVD)

To check the rank of a matrix in python you can use `np.linalg.matrix_rank(X)`. However, we would not do that directly but instead, compute the Singular Value Decomposition (SVD). You have seen in EE1M3 that we can decompose any matrix $\mathbf{X}$ as

$$\mathbf{X} = \mathbf{U}\,\mathbf{\Sigma}\,\mathbf{V}^H\,.$$

Here, $\mathbf{U}$ and $\mathbf{V}$ are *unitary matrices* containing the left and right singular vectors. Since $\mathbf{X}$ has complex entries, we need to extend the results which you know to deal with that. Unitary matrices are the same as orthogonal matrices but use ^{H} (the complex conjugate transpose) rather than ^{T}. Thus,

$$\mathbf{U}^H\mathbf{U} = \mathbf{I}, \qquad \mathbf{V}\mathbf{V}^H = \mathbf{I} \tag{7.2}$$

where $\mathbf{I}$ is an identity matrix. Further, $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values; these are real-valued, positive, and sorted in descending order.

If we write

$$\mathbf{U} = [\mathbf{u}_0, \cdots, \mathbf{u}_{M-1}], \qquad \mathbf{V} = [\mathbf{v}_0, \cdots, \mathbf{v}_{N-1}]$$

then (7.2) shows us that

$$\mathbf{u}_i^H\mathbf{u}_i = 1\,, \qquad \mathbf{u}_i^H\mathbf{u}_j = 0\,.$$

This implies

$$\|\mathbf{u}_i\| = 1\,, \qquad \mathbf{u}_i \perp \mathbf{u}_j\,.$$

Thus, the singular vectors have norm 1 and are orthogonal to each other: we call them orthonormal. The same properties hold for $\mathbf{V}$.

If we denote the diagonal entries of $\mathbf{\Sigma}$ by σ_i, then the SVD can be written as

$$\mathbf{X} = \mathbf{u}_0\,\sigma_0\,\mathbf{v}_0^H\ +\ \cdots\ +\ \mathbf{u}_{M-1}\,\sigma_{M-1}\,\mathbf{v}_{M-1}^H\,.$$

We see that $\mathbf{X}$ is written as a sum of rank-1 components. Since $\|\mathbf{u}_i\| = 1$ and $\|\mathbf{v}_i\| = 1$, we see that the magnitude of each component is determined by σ_i. Therefore, we can use σ_i to judge the number of components, even in the presence of weak signals or noise. E.g., if there are just 2 sources, then σ_0 and σ_1 will be nonzero, and the remaining singular values will be quite small.

7.1.3 Covariance matrix

In array processing, we usually work with the data covariance matrix

$$\mathbf{R}_x = \frac{1}{N}\mathbf{X}\mathbf{X}^H .$$

looking back at the data model, its SVD we had

$$\mathbf{X} = \mathbf{A}\mathbf{S}, \qquad \mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H . \tag{7.3}$$

by inserting these into the covariance matrix we find

$$\mathbf{R}_x = \mathbf{A}\mathbf{R}_s\mathbf{A}^H$$

and also (using $\mathbf{V}^H\mathbf{V} = \mathbf{I}$)

$$\mathbf{R}_x = \frac{1}{N}\mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^H .$$

If we rewrite this as $\mathbf{R}_x = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^H$, with $\mathbf{\Lambda} = \frac{1}{N}\mathbf{\Sigma}^2$ (diagonal), then we see that this is in fact an eigenvalue decomposition of $\mathbf{R}_x$. Thus, the SVD of the (nonsquare) data matrix $\mathbf{X}$ gives the same information as the eigenvalue decomposition of the (square) covariance matrix $\mathbf{R}_x$. The advantage of the SVD is that it always exists, also for nonsquare matrices. The SVD is also numerically more stable to compute.

7.1.4 Assignments

1. Create a function `generate_source(N)` which generates a single random source vector $\mathbf{s}$ of length N samples. This signal should be complex, zero mean, and unit power. Use a function such as `np.random.randn(N)` and use this to generate random real part and random imaginary part of $\mathbf{s}$. Then normalize the output power to ensure it is equal to 1.

2. Create a function `datamodel(M, N , theta0)` which generates the data matrix $\mathbf{X}$. Use the equations in 7.1 as well as your generated functions `generate_source` and `a_lin`

3. Verify the rank of N by computing the SVD of $\mathbf{X}$ and plotting the singular values. Tip: For the SVD, you will need to mind the definition of $\mathbf{V}$:

```
U, S, Vh = np.linalg.svd(X)
V = Vh.conj().T
```

7.1.5 Signal and noise subspace

To introduce super-resolution methods, we first continue the discussion of the SVD of the data matrix $\mathbf{X}$, and focus now on the geometry of the singular vectors. Let's compare the SVD of $\mathbf{X}$ to the data model:

$$\mathbf{X} = \mathbf{A}\mathbf{S}, \qquad \mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H . \tag{7.4}$$

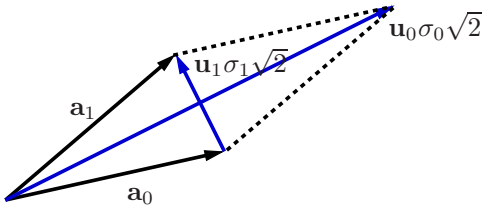


Figure 7.1. Interpretation of the SVD of $\mathbf{X}$.

Comparing these factorizations, it would be tempting to say that $\mathbf{A} = \mathbf{U}$ and $\mathbf{S} = \mathbf{\Sigma}\mathbf{V}^H$. However, this is not the case: the columns of $\mathbf{U}$ are orthonormal, but the columns of $\mathbf{A}$ are not orthogonal at all (and for close angles, they could be rather parallel to each other). Fig. 7.1 shows an interpretation of the situation for two independent equal-powered sources. The dominant singular vector $\mathbf{u}_0$ points into the "common direction" of $\mathbf{a}_0$ and $\mathbf{a}_1$, while the second singular vector $\mathbf{u}_1$ is orthogonal to it and points at their difference. For closely-spaced sources, this difference is small, therefore the 2nd singular value will be small, and more easily masked by noise.

$\mathbf{U}$ is a square $M \times M$ matrix that represent orthonormal directions (or basis vectors) in the data space. The first Q singular vectors in $\mathbf{U}$ correspond to the strongest singular values, which are associated with the signals of interest. These vectors are collected in $\mathbf{U}_s$, which spans the signal subspace, a subspace where the signals from the sources primarily reside. E.g., for 2 sources, it would consist of $[\mathbf{u}_0 \;\; \mathbf{u}_1]$. Likewise, $\mathbf{U}_n$ collect the remaining $M-Q$ singular vectors that correspond to small singular values associated with noise. They span the *noise subspace*. They are orthogonal to the signal subspace. For Q sources, we can split this matrix into two parts:

$$\mathbf{U} = [\mathbf{U}_s \quad \mathbf{U}_n], \qquad \mathbf{U}_s : M \times Q, \quad \mathbf{U}_n : M \times (M-Q).$$

Referring back to Fig. 7.1, we see that the columns of $\mathbf{U}_s$ are an orthonormal basis for the subspace spanned by the columns of $\mathbf{A}$. This property is exploited in several subspace-based direction finding algorithms, in particular MUSIC.

7.2 MUSIC

The acronym MUSIC stands for MUltiple SIgnal Classification [1]. A key property of SVD is that the singular vectors in $\mathbf{U}$ form an orthonormal basis, meaning the signal subspace and the noise subspace are orthogonal to each other. This implies that any vector in the signal subspace is orthogonal to any vector in the noise subspace. This means the array response vectors $\mathbf{a}(\theta)$ which is in the signal space ($\mathbf{X} = \mathbf{AS}$) is orthogonal to the columns of $\mathbf{U}_n$. In particular, the vector function $\mathbf{a}(\theta)$ is orthogonal to $\mathbf{U}_n$ whenever θ is equal to one of the source directions:

$$\mathbf{a}(\theta) \perp \mathbf{U}_n, \qquad \theta \in \{\theta_0, \cdots, \theta_Q\}.$$

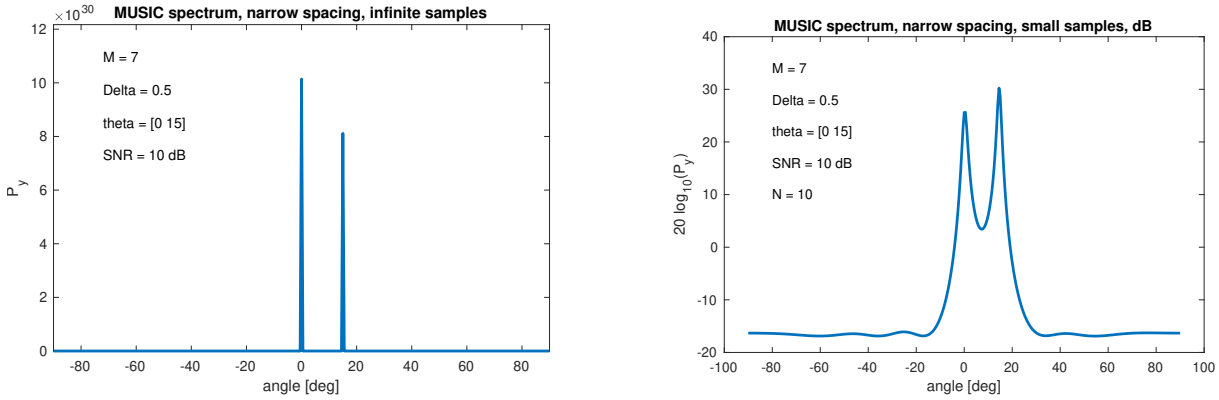


Figure 7.2. Pseudo-spectrum for the MUSIC algorithm.

The idea is to create a "pseudo-spectrum", a function $P_y(\theta)$ which is not quite the output power of the beamformer, but which peaks at directions θ equal to one of the source directions. The choice for MUSIC is

$$P_{\text{music}}(\theta) = \frac{1}{\mathbf{a}^H(\theta)\mathbf{U}_n\mathbf{U}_n^H\mathbf{a}(\theta)}\,, \qquad \theta \in [-\pi/2, \pi/2]\,.$$

We plot this function for θ in its range, and then look for the peaks. At the location of a source, e.g., θ_0, we know that

$$\mathbf{a}(\theta_0) \perp \mathbf{U}_n \quad \Rightarrow \quad \mathbf{U}_n^H\mathbf{a}(\theta_0) = \mathbf{0} \quad \Rightarrow \quad \mathbf{a}(\theta_0)\mathbf{U}_n^H\mathbf{U}_n^H\mathbf{a}(\theta_0) = 0\,.$$

so that the denominator of $P_{\text{music}}(\theta)$ will be zero, and the function peaks. Without noise, or asymptotically for a very large number of samples, the peaks will go to infinity, and we will be able to estimate the directions perfectly.

Fig. 7.2 shows a simulation example, with in (a) the same setting as for Fig. 6.7. It is seen that the spikes are now much more narrow and clear, in fact they go to infinity if you give the algorithm the "true" data covariance matrix (based on expectations, corresponding to infinite number of samples). For a small number of samples $N = 10$, we compute the sample covariance matrix $\hat{\mathbf{R}}_x = (1/N)\mathbf{X}\mathbf{X}^H$, which will deviate from the true $\mathbf{R}_x$. Even so, (b) shows that the spikes are still very high and sharp, and it makes sense to use a dB scale, as shown.

The MUSIC algorithm is considered a *super-resolution* method, meaning the resolution is much higher than that of the matched beamformer. This is possible because more specific knowledge on the data model has been used (the factorization $\mathbf{X} = \mathbf{AS}$, and the structure of the columns of $\mathbf{A}$). Practical problems are that this data model is rather ideal and not quite valid in realistic scenarios.

MUSIC is a well-known classical algorithm. Since its invention (1986), many variants of MUSIC have been proposed, aimed at improving computational complexity, or to resolve problems due to coherent signals (these are signals that are highly correlated, e.g. a source signal and its multipath reflection arriving over a different angle).

7.2.1 Assignments

1. List 5 or 6 reasons why the model $\mathbf{X} = \mathbf{AS}$ is not quite valid in realistic scenarios. (What assumptions were made to obtain this model?)

2. Implement MUSIC as

   ```
   P_music = music(Rx, Q, M, th_range, d, v, f0)
   ```

   (where Q is the number of sources) and test it by recreating Figure 7.2. Note that the python code for generating R_x in this example was already given in assignment 6.4.1.

3. Use your measurements of Module 3, and apply your MUSIC algorithms to see how accurately you can recover the direction-of-arrival of the source. Test the algorithm for both 1 and 2 sources.

Bibliography

[1] R. Schmidt, "Multiple emitter location and signal parameter estimation," *IEEE Tr. Antennas Propagation*, vol. 34, no. 3, pp. 276–280, 1986.

Chapter 8

MID-TERM REPORT

Contents

Now that you know how to acquire and process heart signals, and hopefully have a working MUSIC algorithm for localizing signals, it is time to document what you did so far in a mid-term report. This report will also form the basis of your final report.

Report A mid-term report explaining primarily the signal analysis and localization results. Also answer the specific questions asked in the modules. Suggested length: about 15 to at most 20 pages (excluding the Appendix that lists the Python code).

The report is prepared as a group. The report is graded and contributes to your final grade.

Preparation Completed the preceding Modules (sign-off by the TAs).

Time duration Two homework sessions.

8.1 MID-TERM REPORT

The mid-term report explains your designs with its technical details. The primary focus is to report on your results for the localization and modeling modules, but also the distance sensor measurements should be reported. Max length: 15 to at most 20 pages. Possibly more pages are allowed if some extensions have already been implemented.

Aspects on which the report is judged are:

- *Technical content:* Theoretical justification and accuracy of the results, addressing all requested tasks. Bonus for taking the analysis/design beyond the strict scope of the related Module, in particular for introducing innovative solutions; penalty for unacceptable conceptual mistakes or unfinished work.

  Proof of your results using sufficiently extensive testing.

- *Quality of the submitted report:* Conformity to the requirements of a scientific report (adequate use of equations, figures, citations, cross-references), readability, layout.

  The format is of that of a technical report, and not that of a homework assignment. Thus, the report should contain the results of the various assignments, embedded in a natural (self-contained) way, as these will motivate your solution.

- *Planning* and *teamwork* are also judged. The report should also contain a section that clearly describes these aspects.

The deadline for submission is listed on Brightspace. Submit your report using the corresponding submission folder. Please use a filename that starts with your group number.

The report should be independently readable by a technically skilled committee member who is not familiar with IP3, but you can refer where needed, so don't copy large parts of this manual but summarize the scope in your own words. However the parameters of the algorithms used should be explained in the report.

The modules suggested questions that you can answer in your report —do this in a natural (self-contained) way. You can also refer to additional literature that you consulted. As mentioned, you should try to be concise, and judge yourself what is important to be included.

8.2 HOW TO WRITE AND STRUCTURE YOUR REPORT

Here are some general directions on the structure and contents of your report (mid-term and final report). For the mid-term report, the main focus will be on two sections: heart sound analysis and beamforming. Other chapters such as system integration and conclusions are not yet relevant and can be omitted.

Keep a clear structure and writing style. Make sure you give sufficient factual information (in particular if things don't work). The language should *not* be informal. Keep things concise, "bla-bla" is not appreciated. If you make claims such as "something is the best", first provide evidence (or at least a motivation, or a reference to literature).

Cover: Make sure you specify names, study number, group number, date.

Introduction: First determine who is the reader (in this case, the course coordinator, or evaluating committee member, later in life e.g., your boss). The report should be at his level: find an appropriate balance between context details and conciseness. In general, the introduction of a technical report should present the context and describe the design objectives (problems to be solved), at a sufficiently high level.

After the problem definition/requirements and an initial analysis that identifies the critical design issues, a typical report would split the problem up into sub-problems (subsystems) which are defined in general terms in the Introduction, and developed individually in the subsequent sections.

A picture (block scheme) might be helpful to show the structure and the relations among subsystems. Sometimes, after the introduction a section is needed that describes the background in more technical or mathematical detail, or analyzes the problem in more detail. For the mid-term report, that structure is probably overkill.

In general, an introduction may also contain a literature overview, references to similar work, e.g., how similar problems have been solved and what are the limitations of those solutions (thus motivating your present work), but that is probably not needed for IP3.

The sections on subsystems may follow the same structure but at a more detailed level.

Place your report in context: "This report describes $\cdots$". You don't need to motivate the project (i.e., don't include a text on the relevance of heart monitoring). ChatGPT-generated text is not appreciated.

Sections: Place the results of the Modules each in a separate section. You don't have to repeat everything from a module, but give sufficient context to make your report independently readable. Embed the assignments in a natural way, and give them intelligible names (not "Task 1"). You can combine modules if that makes sense.

For each section, think of the following aspects:

- *Specifications:* What is the objective? What is given already? You may also define notation here.
- *Analysis:* What are the one or two key problems that drive the design? How can these be addressed?
- *Design:* What needs to be designed? What is your approach? Make clear what was already given and what is added by you.
  It is highly appreciated if you include an analysis that explains how accurate your system could be (or will be), in view of hardware limitations. This analysis is then backed up by the verification stage.
- *The resulting design:* This could comment on the implementation, refer to the main variables that you use, etc; generally after reading this, a reader should be able to quickly grasp the Python code in the Appendix.
- **Testing/verification:** How do you test your solution is functional and meets the specifications? What did you measure/observe? Present your results (e.g., measurement results, a Python plot), describe what you see in each plot, and then what you can conclude from this. Don't be naive: things don't work exactly how you design them. Do the debugging systematically. Be sure the plots have correct labels on the axis, and a legend on the line types if you use multiple lines in a single plot. Check the font size; the plot should be readable.
  If something doesn't work, what is your hypothesis on the problem? How would you verify that hypothesis?

It is certainly not appreciated if you only say "It works" or "It didn't work" without documenting this first (i.e., show results/plots/evidence and discuss what is seen).

For IP3, particular emphasis will be placed on the testing aspect. You cannot make claims unless you provide evidence. Your overall system will fail if a subsystem fails, and the purpose of testing is to rule out possible causes of failure.

– *Conclusions:* each section finishes with a conclusion summarizing the results of the Module in a few lines, including claims on expected accuracy. This captures what members of the other sub-group need to know when they use your Module as a black box tool.

Phantom setup and testing: This will be added in the final report. It covers part of the system integration: the initial tests of your system after you bring the various modules together.

Heart sound localization: This continues the system integration and will be added in the final report. Here you further develop your system to be applicable to real data.

Apart from the integration aspects, this is a natural place to describe a GUI, and its functionality. Include a screenshot.

Also in this section, you need a subsection on Verification, now on the complete design (overall test).

Conclusions: Clearly mention if something doesn't work. This is still the formal part of the report, so keep the language sufficiently formal until this point.

Discussion: Here you can include information about the (group) process and any other useful feedback. This part can also include your original planning (work division over team members and time) and the actual outcome. See the Kickoff document: how did that work out? Did everyone contribute?

The material under Discussion can be more informal. In a real (formal) report like a thesis, these sections would be omitted or worked into an Acknowledgement or Postscript.

Appendix: Include Python code and/or other details on your design that may be helpful.

– Structure your code into functions.

– Functions should have a header block that explains what the function does and what the input/output parameters are. Also include author and date information (version, history).

– Describe data structures and other global variables in sufficient detail.

Your appendix has to be sufficiently complete such that the experiments are reproducible by others.

Hint: Review Chapters 1–7 of the manual to ensure all required elements are included.

Chapter **9**

3D LOCALIZATION

Contents

At this point, the subgroup working on M1/M2 has created software to model and generate realistic heart signals, for up to 4 valves. The subgroup working on M3/M4 has beamforming algorithms to localize source directions. However, before integrating your modules, the subgroup working on M3/M4 needs to extend their algorithms so that it can be implemented for three-dimensional heart valve localization.

Learning objectives The following is learned and practiced in this module:

- Extension of beamforming algorithms to 3D localization

Preparation

- Read this module
- Ensure your code from previous Modules is tested and working

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- Your beamforming algorithms from Modules 3 and 4

9.1 EXTENDING YOUR LOCALIZATION ALGORITHMS

In Modules 3 and 4, you created algorithms for the direction finding of narrowband sources in a 1D scenario: The sources were in a single plane far away, and only a single direction is estimated. In comparison to this, we need the following extensions:

- Two angles are needed: beyond azimuth (angle in a horizontal plane) we also need elevation.
- Near-field: sources are close to the array. This has several effects: each microphone sees the source under a different angle, and the attenuation due to path loss is different for each microphone.
- Wideband: we relied on the narrowband condition to be able to translate phase into delay (and then into distance), but it has to be verified if this condition holds.

To resolve the first two aspects, we need to replace the direction vectors $\mathbf{a}(\theta)$ by a direction vector which depends on 3 parameters. Instead of azimuth/elevation/range, it is easier to parametrize the source location in a Cartesian coordinate system as $\mathbf{z} = [x, y, z]^T$, the microphone location as $\mathbf{z}_m = [x_m, y_m, z_m]^T$ ($m = 0, \cdots, M-1$), and to define the direction vector as

$$\mathbf{a}(\mathbf{z}) = \begin{bmatrix} \vdots \\ \dfrac{1}{r_m(\mathbf{z})} e^{-j\omega\tau_m(\mathbf{z})} \\ \vdots \end{bmatrix}, \qquad \text{where } r_m(\mathbf{z}) = \|\mathbf{z} - \mathbf{z}_m\|, \quad \tau_m(\mathbf{z}) = \frac{r_m(\mathbf{z})}{v}. \tag{9.1}$$

Here, $r_m(\mathbf{z})$ is the distance of the source to microphone m, and $\tau_m(\mathbf{z})$ is the corresponding propagation delay. The exponential factor $\exp(-j\omega\tau_m(\mathbf{z}))$ is the corresponding change in phase, assuming the narrowband condition holds, where ω is the center frequency of the band.

The equation also shows a gain factor $1/r_m(\mathbf{z})$ which models the attenuation when propagating over a distance $r_m(\mathbf{z})$. This would be a "free space" attenuation which obviously will not be the case in our application, so this is just a simple approximation of the effect.

As coordinate system, use the one proposed in Sec. 5.3, where x, y is horizontal/vertical (on a standing person), and z is depth (into the body). As origin of the coordinate system, it is customary to take the center of the microphone array, but you could take any other convenient reference point.

In comparison to Module 3, we now replace $\mathbf{a}(\theta)$ of Eqn. (6.1) by the new definition (9.1), in all expressions where it appears. Thus, for our data model with V sources (valves) at location $\mathbf{z}^v$, we have

$$\mathbf{x}(t) = \sum_{v=1}^{V} \mathbf{a}(\mathbf{z}^v)\, s_v(t).$$

Similarly, for the matched filter, we look at the output power of the beamformer while we scan all potential source locations $\mathbf{z}$:

$$P_{mf}(\mathbf{z}) = \mathbf{a}^H(\mathbf{z})\mathbf{R}_x\mathbf{a}(\mathbf{z}).$$

For MVDR, we scan

$$P_{mvdr}(\mathbf{z}) = \frac{1}{\mathbf{a}^H(\mathbf{z})\mathbf{R}_x^{-1}\mathbf{a}(\mathbf{z})},$$

and for MUSIC, we scan

$$P_{music}(\mathbf{z}) = \frac{1}{\mathbf{a}^H(\mathbf{z})\mathbf{U}_n\mathbf{U}_n^H\mathbf{a}(\mathbf{z})}.$$

To make a scan, we vary $\mathbf{z} = [x, y, z]^T$ in small steps, e.g., 1 cm or less, and compute the corresponding output power $P(\mathbf{z})$ of each of the beamformers. In theory, to inspect the resulting data, you would need to create a 4D plot (x, y, z, P). Practically, you can create a series of 2D images (x, y) with color determined by P, and vary the depth z over a small range of interest. Each image then shows a slice of the situation at the selected depth.

9.2 ASSIGNMENTS

1. Modify your algorithms and create new function to use the new definition of $\mathbf{a}(\mathbf{z})$, and to create a series of images while $\mathbf{z}$ is scanned in small steps.

   ```
   a_z(s_position, mic_positions, M, v, f0)
   music_z(Rx, Q, M, xyz_points, v, f0)
   mvdr_z(Rx, M, xyz_points, v, f0)
   ```

   where `xyz_points` is a 2D array of (number of points $\times 3$) that includes the coordinates of all the points that you want to scan in the search space to localize the source.

2. Create a simulation setup to generate model data (random noise), where you specify true source locations and microphone locations. As source signals, use white noise (independent for each source). Start with 1 source in front of the array, at a depth of about 15 cm. Test your algorithms on this modeled data. Vary its (x, y) location. Create a series of images while $\mathbf{z}$ is scanned in small steps. Module 2 has already made this model, so with some small modifications this function can be used for this task.

3. Use the modified version based on $\mathbf{a}(\mathbf{z})$ to determine the direction of arrival (DoA) of the source in the linear array experiment described in Section 6.6 for some of the cases.

   Hint: Although you are using your 3D localization algorithm for this experiment. We are still only interested in finding the direction of the source, not its position in 3D space. Therefore, instead of scanning the entire three-dimensional space, which is very inefficient, focus on the points located along the perimeter of a circle with a known radius of 8 meters around the center of the array. First convert the locations of these points with $r = 8$m and θ ranging from -90° to 90° into their corresponding XYZ coordinates and scan them using the MUSIC or MVDR algorithm. Compare different values for bandwidth and center frequencies for best DoA estimation. Document all your results and values.

   If your test works, go to 2 sources.

4. Create good quality plots for your report. Provide comments on the accuracy of the algorithms.

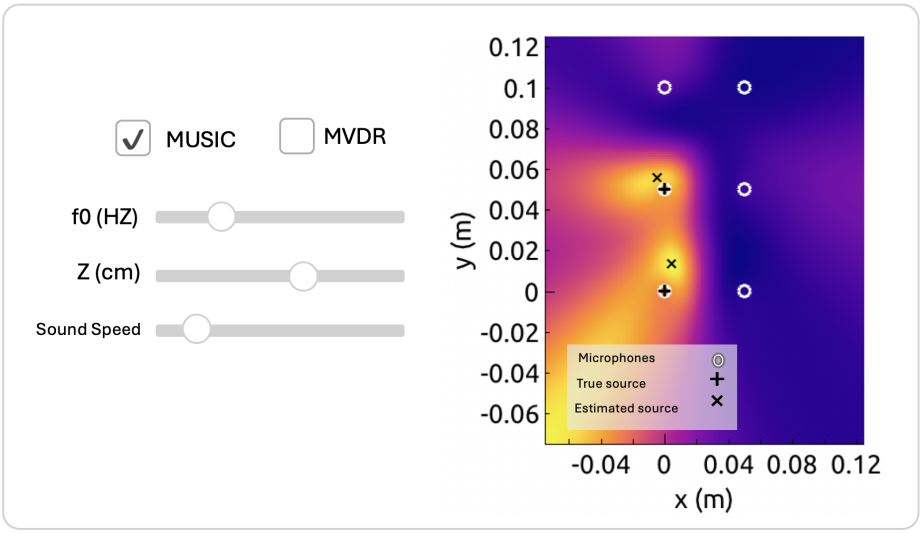


Figure 9.1. Example GUI for visualization and parameter adjustment.

9.3 VISUALIZATION AND GUI DESIGN

When working on improving and optimizing your algorithm, you will need to adjust different parameters and check the outputs. This process usually involves a lot of trial and error, and one of the best ways to evaluate your results is through visual inspection. To make this easier as an strongly recommended extension, you can create a graphical user interface (GUI). With a GUI, you can quickly change parameters such as center frequency, bandwidth, z-plane range, speed of sound, etc., and immediately see how the results are updated. You can also include the final visualization in your report to demonstrate how well your algorithm performs. For clarity, make sure your visualization shows the microphone locations, the true source location, and the estimated source location. Figure 9.1 shows an example to give you an idea of how this might look.

9.4 EXTENSIONS

There are additional opportunities to extend and enhance your work. Below are some of these options. We strongly recommend discussing these possibilities with the course instructor, especially if you are experimenting with ideas from research papers or developing your own design. This will help ensure the success of your work and prevent unnecessary or overly complex efforts. Please note that the grade you receive will depend on the depth, scope, and performance of your algorithms, rather than the number of extensions you choose to implement.

General extensions: You can further enhance all previous algorithms beyond the manual by exploring the following areas:

- *Microphone array improvements:* Experiment with new and more efficient microphone array structures to enhance signal capture and localization.
- *Frequency bin selection:* Optimize the selection of frequency bins to focus on the most relevant frequency ranges for heart sound analysis.

Murmur detection using source localization (multi-channel, simulation): In a normal heart sound, you should detect two valves (Mitral and Tricuspid) during S1, and two other valves (Aortic and Pulmonic) during S2, with silence in between during systole and diastole. In an abnormal heart sound, murmurs may be audible during the normally silent phases. Some common heart abnormalities include:

- Mitral Valve Regurgitation
- Mitral Valve Stenosis
- Aortic Valve Regurgitation
- Aortic Valve Stenosis
- Ventricular Septal Defect (VSD)

Real multi-channel data from these patients is unavailable. Therefore, you will rely on your teammates who completed Module 2. They have to simulate the data for you as describen in Section 5.5. You will then attempt to detect the sources of sound in the silent phase. By analyzing the audio during S1 and S2, you can identify the signal subspace, determine the number of sources, and localize them. For the first four arrhythmias (M or A valve abnormalities), check if the sources correspond to expectations. For VSD, an additional source should be detected during systole.

Source separation: As a more advanced task, you can focus on source separation to extract individual valve signals from the cardiac activity, which could be a valuable outcome of this work. Once the location of the sources are identified using MUSIC, the sources can be separated by focusing on the specific directions/locations associated with each heart valve sound. However, this is a very advanced application. To start, we advice you to try it with recordings from the linear array.

Chapter **10**

INTEGRATION AND TESTING

Contents

At this point, the subgroup working on M1/M2 has created software to model and generate realistic heart signals, for up to 4 valves. The subgroup working on M3/M4 has beamforming algorithms to localize source location in 3D. With these working subsystems in place, it is now time to put these together and test the beamforming algorithms on realistic heart sounds!

Learning objectives The following is learned and practiced in this module:

- System integration and systematic testing

Preparation

- Read this module
- Ensure your code from previous Modules is tested and working

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- Your modeling software of Module 2
- Your beamforming algorithms for 3D Localization
- Hardware: Phantom heart model, PreSonus soundcard, amplifier card, 6-element stethoscope array
- Raspberry Pi 5 loaded with the IP-3 Recording Tool
- Sound source signals (in .wav format) to play on the loudspeakers (white noise)

10.1 TEST ON FULLY SIMULATED RECORDINGS

Now that you have working localization algorithms also for 3D positions, it is time to combine this with the 3D cardiac model developed in module 3. That model can generate realistic data that replaces the test data of the previous section.

10.1.1 Assignments

Make that integration! You will also have to augment your data processing: first preprocess and then segment the received signals in time domain. Use your annotations into S1/S2 to first collect a data matrix $\mathbf{X}_1$ containing only samples from S1, and then a second data matrix $\mathbf{X}_2$ with samples from S2. Construct corresponding data covariance matrices $\hat{\mathbf{R}}_1$ and $\hat{\mathbf{R}}_2$. Each should contain the contributions of only two of the 4 valves. (Use the SVD to verify that the rank of each covariance matrix is 2.)

Before starting with localization, take time to clearly specify what you need from Module 1 and 2. Do you require valve signals, signals recorded by microphones, or adjustments to certain parameters to test how your algorithm performs? Should you work with raw segmented signals, or is preprocessing necessary—and if so, could these steps interfere with your application? Always double-check that the data you are using matches your actual requirements, as this is often where mistakes occur due to unclear communication.

Test your algorithms: Assess whether your approach accurately recovers the positions of the two valves active during S1, followed by the two valves active during S2. Experiment with various values for Δf and `f0` to determine the optimal settings for Direction-of-Arrival (DoA) estimation. Since each valve sound is simulated using distinct frequencies, certain frequency bins may prove more effective for localizing specific valves. Identify the values that yield the most accurate results for both sources and provide a comparative analysis.

Note: The optimal results observed here are based on the simulated sound data, which may not directly apply to real heart sound recordings.

10.2 RECORD AND TEST ON PHANTOM MODEL DATA

If all the above works, we are ready to test on real microphone array data. Although the ultimate goal is to apply this to human data, such data is sufficiently hard to work with, because

- There is no ground truth: we only know there are 4 valves, but not exactly where they are positioned. So if the algorithm produces some results, we can only say if it looks plausible.
- Human bodies produce a lot more sounds than just in the heart; e.g., the lungs, the intestines. Microphones pick up a lot of environment noise as well.
- The propagation inside the body is a lot more complicated than the free space “line of sight” propagation modeled by us. E.g., different materials have different propagation speeds, and there

will be multipath effects.

As a compromise, we will first test on a dummy phantom model. The model was shown in Sec. 3.4. It consists of two "valves" each consisting of a loudspeaker. Thus, we could transmit either two S1 valve signals or two S2 valve signals on these.

10.2.1 Assignments

1. Make test recordings using the phantom model. Use the PreSonus sound card, Raspberry Pi 5 with IP-3 Recording Tool, the amplifier card to connect to the internal loudspeakers in the phantom model, and the 6-element stethoscope array.
    - One source (each loudspeaker separately), playing white noise.
    - Two sources simultaneously, playing white noise.
    - One source, playing one single valve (works better with higher frequencies)
    - Two sources simultaneously, playing S1 or S2 model signals. (works better with higher frequencies with larger difference)
2. Also select a coordinate system and document the true locations of the valves and the microphone array.
3. Process your recordings using your algorithms. You will also have to estimate the propagation speed first! For this, use the one-source recording and the known distance.
4. Make good quality plots for your report, and comment on the accuracy that you obtained: are you able to locate the two valve signals?

Note that, ideally, we would play real heart signals from data repositories. However, individual valve signals (recorded *at* the valve location) are obviously unavailable: how could these ever be acquired?

Chapter **11**

FINAL CHALLENGE AND FINAL REPORT

Contents

You have made it to the final part: the Final Challenge. At this moment you have a working system, tested on phantom data. How would your system perform on real cardiac data? It is time to find out!

Learning objectives The following is learned and practiced in this module:

- Testing your algorithms on real data

Preparation

- Read this module
- Ensure your code from previous Modules is tested and working

Time duration Two lab sessions and two preparation/reporting sessions at home

What is needed

- Laptop/PC running Python
- Challenge test recordings on Brightspace
- PreSonus soundcard, amplifier card, 6-element stethoscope array
- Prepared Raspberry Pi 5 with IP-3 Recording Tool
- Willing test subject

11.1 INTRODUCTION

The final assignment in this project is a very challenging one: use the 6-element stethoscope array to localize the heart valves. This is not easy, because the sound wavelength is much larger than the distance between the valves.

Ideally, at this point we would ask you to don a white coat, take your stethoscope array, convince a willing subject to lie down, and acquire the data. However, this is tricky for more than one reason.

- Measurement conditions in the Tellegenhall are less than ideal. It is probably very noisy and a comfortable couch is missing. Also, several groups are competing for the same measurement setup.
- There are some ethical concerns that need to be considered before human data can be collected.
- For comparing groups, it is more fair if they all have identical conditions.

To avoid these issues, we will give you, on Brightspace, a file with prerecorded data (including some meta-data). Test your algorithms on this dataset and report the results!

Do not postpone to document your work. The final report is needed very shortly after the demonstrations.

11.2 ETHICAL ASPECTS

Informed consent

Although stethoscope data is not considered invasive, initially it is still "personal medical data" and hence there are some issues to consider.

- Consent: under which conditions are you allowed to acquire data? Is signing an "informed consent" form sufficient, and what should be the content of that form?
- Privacy: is it sufficient to synonymize the data before using/sharing/publishing it?
- Medical ethical committee: before hospitals can carry out medical experiments involving patients, they need to pass by the Medical Ethics Committee (METC) or, more lightly, the Human Research Ethics committee (HREC). TU Delft has such an HREC committee. Is that relevant here, and should that be done by the practicum coordinators, or by each individual group?

Before proceeding with data acquisition, you need to have a discussion within your team about the recording process and prepare an informed consent form. Even though you will record your own heart signals and not those of external participants, all groups are required to prepare an informed consent form. This exercise is designed to familiarize you with the challenges of recording data from human participants.

The consent form should include the following sections: Title of the study, Researchers' names, Purpose of the study, Description of your (test subject) participation, Methods used to analyze the data, Potential risks, Potential benefits, Participants' rights, Confidentiality (how it is stored), Contact information, Access to research results, Consent and signature.

Examples of informed consent forms can be found online to guide you. The document should be a maximum of two A4 pages. Note that the consent forms will not be reviewed or approved by the Human Research Ethics Committee (HREC), as this is an educational exercise, not a formal research study.

Before you do any recordings, the volunteers should sign the form, and you upload the form in the designated folder on Brightspace (under Assignments).

Ethics workshop

You have learned/will learn about ethical dilemmas in an engineering practice in an ITAV lecture. The learning objective of this introduction is to train you to evaluate ethical dilemmas in engineering practice (using ethical theories and approaches). You will practice this by identifying and analyzing an ethical dilemma that is applicable to your project.

In this group assignment, you will apply the Ethical Cycle to analyze an ethical dilemma related to your project and reflect on the process. In your final report, you will add a chapter (1 page) discussing this dilemma, and summarizing how two ethical perspectives would evaluate this dilemma. You then recommend your preferred solution to the dilemma.

Some preparation is needed before the ITAV lecture. Consult Brightspace for the details.

11.3 HUMAN DATA ACQUISITION AND FINAL ALGORITHM TESTING

You have the opportunity to record your own heart signals. For this, we have provided a private room equipped with curtains and a bed. This setup allows you to lie down and place the array on your chest. For better signal quality, it is recommended to shave the area, remain as still as possible, and minimize clothing movement, as small movements can introduce noise. While you can keep your shirt on, be mindful of the potential noise caused by fabric movement. However, you do not need to take off your shirt necessarily.

We encourage you to perform your own recordings for the full experience. To do this, at least one team member will need to assist with the recording process. However, we understand that not all group members may feel comfortable recording their own data in this setting. If this is the case, please inform us, and we will provide pre-recorded data for your use. It is important to discuss this matter openly within your team and ensure that no peer pressure is applied to convince others to participate in recording if they are uncomfortable.

Assuming the ethical situation is cleared up and your informed consent form is prepared, find one or two willing volunteers in you group and acquire data! Use the 6-element array. One good recording is

sufficient. But you can have multiple ones in the given time slot. Record several heart beats (10 to 20 seconds) such that later you can remove unclean data. While recording, also document the positions of the array elements with reference to anatomical markers (e.g., the sternum and intercostal spaces; see Sec. 2.3). When storing files, immediately make it anonymous, although other measurement settings must be documented.

The end goal of the project is to show a stack of images which clearly shows the valve locations and whether they are active during S1 or S2; and next, if the estimated locations are anatomically plausible. Given the poor resolution due to long wavelengths, this is not at all easy! Even if you are not successful, you can still make meaningful observations, such as on the rank of the covariance matrices (showing the number of active sources).

11.4 FINAL CHALLENGE

The final challenge is this: on the indicated date (last labday) you show and informally discuss your results in this chapter to the practicum coordinators. This is not giving a grade, but the feedback might help you while finishing your report.

11.5 FINAL REPORT

Instructions for the final report are similar to those of the midterm report (see Chapter 8). Aim for a **well-structured**, compact yet complete report of about 30 pages (plus Python code in an appendix). Do not forget to systematically report on testing/verification: how do you test (each subsystem, and the entire system), what are the results from the test, what do you conclude. Include the results of the final challenge in the report as well. Note that the final challenge is not a test, rather it is a demonstration. With extensive testing, these results won't be a surprise to the reader!

The report is judged by committee members that are not indepth familiar with IP3, or the manual. Your report has to be sufficiently self-contained.

The submission deadline is listed on Brightspace, typically it is one day after the final challenge. Submit your report using the corresponding submission folder.

In comparison to the structure defined for the midterm report in Chapter 8.2, two additions are needed which document the ITAV sessions.

Ethics After the Conclusions chapter, insert a chapter related to the ITAV Ethics assignment: discussion on a moral dilemma related to your project, seen through two ethical lenses. See the template structure on Brightspace (under Content/ITAV Ethics) for details.

Teamwork using Scrum As part of the Discussion chapter, insert a section on teamwork where you reflect on aspects related to the Scrum workshop. Which aspects did you try, and how did that work out?

If you did not implement some aspects, then how did you assure that all team members were efficiently contributing and you would reach your goals in time?

Describe two situations in which you and your team benefitted from Scrum, or how you and your team could have benefitted from Scrum if implemented more thoroughly. For each situation, reflect on the following aspects:

- *Situation:* Describe the situation and provide some context (who, what, where, when).
- *Task:* Describe your role or task.
- *Action:* Examine your behavior. What did you do and why? What were your thoughts?
- *Result:* Describe the result of your action. What was the effect on the team and on the project.
- *Reflection:* Are you satisfied with what happened? Why or why not? Did your actions align to the Scrum methodology and the Scrum values? Did you at that moment consider if any Scrum rules were applicable?
- *Advice:* Give yourself a piece of advice. What would you do next time?

11.6 FINAL PRESENTATION AND DISCUSSION

In week 10 (consult Brightspace for the exact date), you present and defend your final report in front of an examination committee. The examinators will ask questions about your design choices and aspects of teamwork. This will be part of your grade.

The presentation lasts at most 10 min. Focus on the highlights and special features of the design, and also mention the work breakdown and distribution of tasks to team members.

The examination will last about 30 min. After the examination you will be asked to fill in a peer review form. Individual grades are differentiated depending on staff observations and the outcome of the peer review.

Appendix ***A***

ARRAY SIGNAL PROCESSING CONCEPTS

Contents

A.1 NARROWBAND CONDITION

We would like to use more general (analytic) signals than a sinusoid. We can take a sum of sinusoids with frequencies that are close together. They don't need to all have the same amplitude. We can write this as

$$s(t) = \int S(\Omega) e^{j\Omega t} \mathrm{d}\Omega \,.$$

This is indeed a sum of sinusoids, each with their own amplitude $S(\Omega)$. See Fig. A.1. (Of course, this just shows an interpretation of the Inverse Fourier Transform.) Then

$$s(t-\tau) = \int S(\Omega) e^{j\Omega(t-\tau)} \mathrm{d}\Omega = \int e^{-j\Omega\tau} S(\Omega) e^{j\Omega t} \mathrm{d}\Omega$$

If $\Omega\tau \approx \Omega_0\tau$ for all Ω in the support of $S(\Omega)$, then

$$s(t-\tau) = e^{-j\Omega_0\tau} \int S(\Omega) e^{j\Omega t} \mathrm{d}\Omega = e^{-j\Omega_0\tau} s(t)$$

If the support of $S(\Omega)$ is the interval $[\Omega_0 - B/2, \Omega_0 + B/2]$, then the condition translates to

$$B\tau \ll 2\pi$$

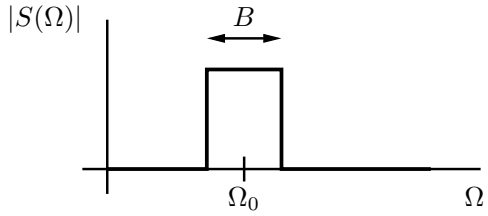


Figure A.1. A narrowband signal.

This is called the *narrowband condition*. The factor 2π is such that it disappears if B is expressed in Hz, as we usually do. Here, the worst-case τ is the maximal delay that you can expect over the array, which occurs when the signal arrives at $\theta = \pm\pi/2$. Let $D = (M-1)d$ be the size of the array (the maximal baseline). Then we need

$$B\,[\text{Hz}] \ll \frac{v}{D}$$

If the narrowband condition is not satisfied (which quickly happens for audio signals), then we need to resort to *wideband array processing*. The usual solution is to apply a bandpass filter, such that the output of the bandpass satisfies the narrowband condition. We then vary the center frequency of the bandpass over a range of values. A convenient way to do this is to apply the Short-term Fourier Transform (STFT), which is your spectrogram, but without taking the absolute values.

A.2 MICROPHONE SPACING

The microphone array samples in space, and just as with sampling in time, the issue of aliasing comes up. The array response vector has entries of the form $\exp(-j(d/v)\sin(\theta)\Omega_0)$, and our direction finding algorithms are first going to estimate the phase, $(d/v)\sin(\theta)\Omega_0$. To be able to estimate θ, we need a one-to-one relation between θ and $(d/v)\sin(\theta)\Omega_0$, for $-\pi/2 \le \theta \le \pi/2$. Therefore, we need to require

$$-\pi \le (d/v)\sin(\theta)\Omega_0 \le \pi, \qquad \theta \in [-\pi/2, \pi/2]\,.$$

Hence

$$\frac{d}{v}\,\Omega_0 \le \pi$$

Inserting $\Omega_0 = 2\pi F_0$ and the wavelength $\lambda = v/F_0$ gives

$$d \le \frac{1}{2}\lambda\,.$$

Thus, to avoid spatial aliasing, we have to sample more densely in space than half the wavelength. Alternatively, if we define $\Delta = d/\lambda$ as the microphone spacing expressed in wavelengths, we need $\Delta < 1/2$.

A lot of array processing literature makes a standard assumption of half wavelength spacing; however, this is certainly not always practical. During wideband processing, if we have a filterbank with several center frequencies, then Ω_0 varies, and therefore also λ varies.