EEE3097S 2023 MILESTONE 1 REPORT

ACOUSTIC TRIANGULATION USING MULTILATERATION AND GENERIC CROSS-CORRELATION WITH PHASE TRANSFORM

18 August 2023

Simon Carthew - CRTSIM008
Robert Dugmore - DGMROB001
Si Teng Wu - WXXSIT001

INTRODUCTION

This project aims to implement an acoustic triangulation system on a small scale in a controlled environment, using relatively cheap hardware.

Higher-level uses of acoustic triangulation can help with tracking and locating various sound sources such as vehicles, gunshots, human speakers, weather phenomena, and more.

Though this project is not directly applicable to the higher-level use cases, it can serve as a useful investigation of an implementation of acoustic triangulation system and some of the issues associated with this implementation.

TABLE OF INDIVIDUAL CONTRIBUTIONS

All sections of this report were extensively discussed by the entire project team. After this, individual sections were allocated for this report to the various team members, as follows:

Name	Sections
Simon Carthew	OP Diagram, Pi Synchronisation, Hardware, Project timeline
Robert Dugmore	High-level requirements, feasibility and risk analysis, SWOT analysis, Signal Acquisition, User Interface, Trello
Si Teng Wu	Time Delay Estimation, Triangulation, Acceptable performance definitions, System tests

HIGH-LEVEL REQUIREMENTS AND IMPLEMENTATION DESCRIPTION

Core requirements from project brief:

- Implement an acoustic triangulation system
- Accurately locate the position of a sound source
- Use time-difference of arrival (TDoA)
- Rectangular grid, two dimensional coordinates
- Use two Raspberry Pi microcontrollers and four microphone breakout boards
- Display results using a graphical user interface

Additional requirements (bonus marks):

- Implement acoustic source tracking (continuous tracking of the sound source)
- Adapt all relevant parameters to be suitable for continuous tracking

General description of our implementation:

Each Raspberry Pi is connected to 2 of the microphones using the I2S protocol. The Raspberry Pis are set to record data on the falling edge of a shared clock, ensuring synchronisation between the systems. A calibration procedure will account for any fixed delays in this process.

Once data has been read by the Raspberry Pis, it will be transmitted to a laptop over a USB serial connection. The laptop will handle TDoA calculations, triangulation, and the user interface.

Code will be written using Python and MATLAB coding languages. This offers the simplest solutions, as between the two languages there are many existing code libraries which can assist with various sections of the project.

The system will have soft-configurable parameters such as sampling frequencies, and start and stop times for data recording.

The primary priority will be to implement a functional system which meets the minimum specifications of the requirements above. Thereafter, any additional time available will be spent on the following (in order of priority):

- Implementation of continuous tracking
- Enhanced system accuracy
- Enhanced user interface and user experience

The project will be implemented using Multilateration and Generalised Cross-Correlation with Phase Transform (GCC-PHAT) algorithms.

Multilateration uses the Time-Difference of Arrival of a sound between 2 sensors to trace a hyperbola of possible locations of the sound source. By using multiple pairs of microphones to trace multiple hyperbola, an intersection can be found which indicates the exact position of the sound source.

Alternative methods of acoustic localisation:

- Trilateration. Trilateration uses the absolute difference between the time of departure of a sound and the time of arrival at a microphone to determine a distance between the sound source and the microphone. Repeating this process with multiple microphones allows multiple circles with radius of the absolute distances to be traced and compared to determine a precise location of the sound source. This is not feasible for this project, since the hardware setup does not allow the time of the departure of the sound source to be known [1].
- Binaural recording. This uses a model of a human head with microphones in the ears to determine the location of a sound source using various factors of the sound in both the time and frequency domains. This is not feasible for our implementation since we do not have the apparatus needed [1].
- Beamforming. Beamforming uses time differences between an array of microphones to compute the locations of sounds, typically at large distances. Due to the limited number of microphones and small distances in the project requirements, this is not feasible for our implementation [1].

Use of fourth microphone:

- Sound tracking technically only requires 3 microphones. We have chosen to use the fourth microphone to enhance the accuracy of the system, by tracing and comparing additional parabolas in the plane.

Alternatives considered for use of fourth microphone:

- Using the fourth microphone for 3D sound tracking. This was rejected because the project-brief places an emphasis on accuracy and only specifies 2D sound tracking.
- Only measuring the TDoA between the 2 pairs of 2 microphones, to make implementation easier. We rejected this idea because we are prioritising accuracy over the minimally easier implementation that this would entail.

Sub-systems:

The project has been divided into 6 subsystems:

- Pi synchronisation
- Signal acquisition

- TDoA calculation
- Triangulation
- User interface
- Hardware implementation

More detailed requirements are described for each subsystem, alongside specifications, intra-subsystem interaction, and inter-subsystem interactions.

FEASIBILITY ANALYSIS, RISK ANALYSIS, AND POSSIBLE BOTTLENECKS

Feasibility:

The core requirements for multilateration are met by the available hardware (3 microphones, placed throughout a 2D plane). Additionally, the fourth microphone allows for implementation of further measures to improve the project.

Raspberry Pis are well documented online and support many useful interactions with hardware through the 40-pin GPIO header, including generation of clock signals and reading of interrupts. This makes the interaction between hardware and software easy to manage.

Finally, techniques for multilateration are well documented online, meaning that existing knowledge of acoustic localisation can be applied to this project.

Overall, the available hardware, software, and information resources mean that adequate implementation of this project is feasible.

Risks/bottlenecks:

Inadequate signal-to-noise ratio for feasible analysis of audio data. If necessary, this can be circumvented by testing the system in an environment with low noise levels, or by tracking and filtering frequencies that are not common in the background noise.

Inadequate time correlation between sound data on the two Raspberry Pis. This can be dealt with using a calibration procedure to determine a fixed delay between the times data read on the two Raspberry Pis, or by reverting to the “easier” implementation method described above which only compares the time difference of arrival of the two microphones on each individual Raspberry Pi.

Inadequate serial transfer speeds from the Raspberry Pis to the laptop. If necessary, some processing can be performed on the Raspberry Pis before data is sent to the laptop, or the sampling frequency and/or audio quality can be reduced to allow data to be sent at a slow transfer speed.

PROJECT PLANNING SWOT ANALYSIS

Strengths:

- Team members are willing to communicate, collaborate, and work towards to the project
- Team members are willing to work hard to ensure a successful project
- At a 3rd year level, team members have sufficient education and experience to implement a project of this nature

Weaknesses:

- Team members have busy schedules which sometime clash with working on the project

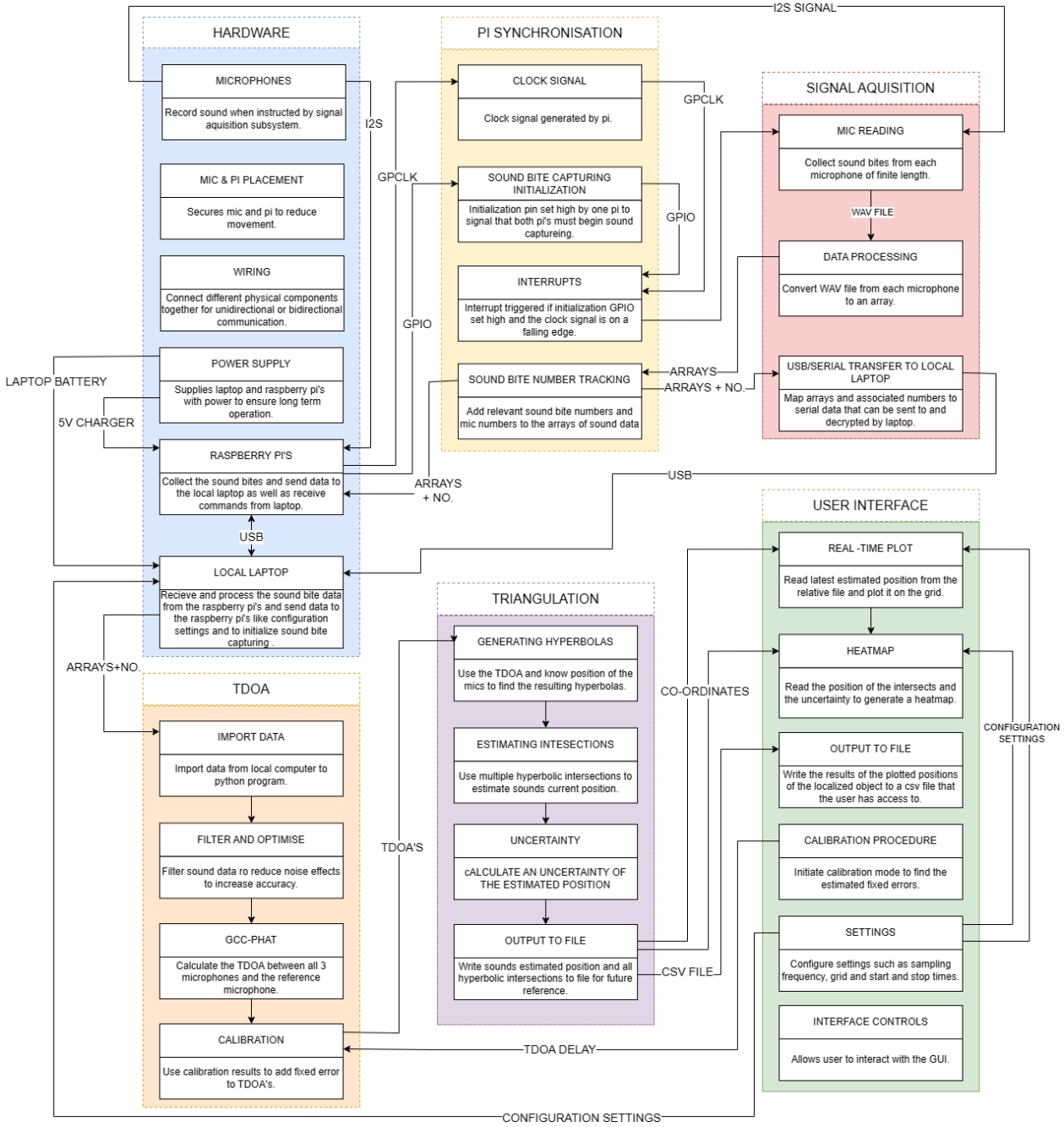
Opportunities:

- Extensive online papers detailed implementation of similar projects
- Existing code libraries which implement aspects of the project
- Individuals who are willing to assist the team, such as the EEE3097S course Teaching Assistant

Threats:

- Limited hardware resources and budget available if current hardware proves inadequate for project implementation
- Online code libraries often require adaptation to meet the specific needs of the project

OP DIAGRAM


I2S SIGNAL
HARDWARE
MICROPHONES
Record sound when instructed by signal aquisition subsystem.
MIC & PI PLACEMENT
Secures mic and pi to reduce movement.
WIRING
Connect different physical components together for unidirectional or bidirectional communication.
POWER SUPPLY
Supplies laptop and raspberry pi's with power to ensure long term operation.
LAPTOP BATTERY
5V CHARGER
RASPBERRY PI'S
Collect the sound bites and send data to the local laptop as well as receive commands from laptop.
USB
LOCAL LAPTOP
Recieve and process the sound bite data from the raspberry pi's and send data to the raspberry pi's like configuration settings and to initialize sound bite capturing .
ARRAYS+NO.
I2S
GPCLK
GPIO
ARRAYS + NO.
PI SYNCHRONISATION
CLOCK SIGNAL
Clock signal generated by pi.
SOUND BITE CAPTURING INITIALIZATION
Initialization pin set high by one pi to signal that both pi's must begin sound captureing.
INTERRUPTS
Interrupt triggered if initialization GPIO set high and the clock signal is on a falling edge.
SOUND BITE NUMBER TRACKING
Add relevant sound bite numbers and mic numbers to the arrays of sound data
GPCLK
GPIO
ARRAYS
ARRAYS + NO.
SIGNAL AQUISITION
MIC READING
Collect sound bites from each microphone of finite length.
WAV FILE
DATA PROCESSING
Convert WAV file from each microphone to an array.
USB/SERIAL TRANSFER TO LOCAL LAPTOP
Map arrays and associated numbers to serial data that can be sent to and decrypted by laptop.
USB
USER INTERFACE
REAL -TIME PLOT
Read latest estimated position from the relative file and plot it on the grid.
HEATMAP
Read the position of the intersects and the uncertainty to generate a heatmap.
OUTPUT TO FILE
Write the results of the plotted positions of the localized object to a csv file that the user has access to.
CALIBRATION PROCEDURE
Initiate calibration mode to find the estimated fixed errors.
SETTINGS
Configure settings such as sampling frequency, grid and start and stop times.
INTERFACE CONTROLS
Allows user to interact with the GUI.
CONFIGURATION SETTINGS
TRIANGULATION
GENERATING HYPERBOLAS
Use the TDOA and know position of the mics to find the resulting hyperbolas.
ESTIMATING INTESECTIONS
Use multiple hyperbolic intersections to estimate sounds current position.
UNCERTAINTY
cALCULATE AN UNCERTAINTY OF THE ESTIMATED POSITION
OUTPUT TO FILE
Write sounds estimated position and all hyperbolic intersections to file for future reference.
CO-ORDINATES
CSV FILE
TDOA
IMPORT DATA
Import data from local computer to python program.
FILTER AND OPTIMISE
Filter sound data ro reduce noise effects to increase accuracy.
GCC-PHAT
Calculate the TDOA between all 3 microphones and the reference microphone.
CALIBRATION
Use calibration results to add fixed error to TDOA's.
TDOA'S
TDOA DELAY
CONFIGURATION SETTINGS

PI SYNCHRONISATION

Brief description:

This subsystem is responsible for ensuring that all the sound bites captured by the four microphones are captured at the same time and that the sound bites received by the local computer all correlate to the same time of capture. A clock signal is outputted by the master pi and into a GPIO pin on both itself and the secondary pi. The falling edge of the clock signal is used to synchronise the capturing of sound bites by all four microphones. The sound bites are numbered and sent to the local computer as arrays using the usb ports on both raspberry pi's.

Alternatives Considered:

An alternative method of synchronisation that was considered was using the network time protocol (NTP) to sink the "real-time" clocks of both the raspberry pi's. The idea was to instruct both raspberry pi's to capture sound bites at the designated "real-times" and because their clocks are synchronised, the four sound bites captured would all be captured simultaneously.

After careful consideration, it was decided that this concept may not be feasible. The issue with it is that NTP can have inaccuracies of up to three milliseconds which would create delays between sound bite capturing. This delay would result in incorrect TDOA values and localization.

REQUIREMENTS	
Clock Signal	A clock signal must be used as a trigger for both raspberry pi's to capture sound bites. The frequency must be fast enough for real time tracking but also not faster than the how long it takes for each sound bite to be processed.
Sound Bite Capturing Initializing	Both pi's must begin sound bite capturing on the same clock cycle.
Interrupts	An interrupt must be used on both pi's to take a sound bite on all mic's that is triggered if it is a falling edge of the clock signal and sound bite capturing has been initialised.
Sound Bite Number Tracking	The number and microphone that captured a sound bite must be kept track of.
SPECIFICATIONS	
Clock Signal	The clock signal must be generated by the GPCLK pin on the master pi at 10 Hz and must be connected to the GPIO5 pins on both pi's.

Sound Bite Capturing	The local laptop must send commands via USB to master pi that instruct the pi when sound capturing must begin and end.
	The master pi's GPIO4 pin must be set high when sound capturing must begin and low when it must end.
	The GPIO4 pin of the master pi must be connected to the GPIO 4 pin of the secondary pi. The secondary pi must begin sound capture when GPIO4 is high and must stop when it is low.
Interrupts	The interrupt must be triggered on both pi's when GPIO4 is set high and the clock signal connected to GPIO5 is on a falling edge
	When triggered the interrupt must call upon the signal acquisition sub module to capture the sound bites.
Sound Bite Number Tracking	The number of the sound bite captured and which microphone captured the sound bite must be transmitted before its associated sound bite is transmitted in the form <SoundBiteNumber_MicNumber> using USB.
INTRA-SUB INTERACTIONS	
Sound Bite Capturing - Interrupts	The interrupts to capture a soundbite must only be triggered when the GPIO4 pin is set high by the first pi.
Clock signal - Interrupts	The interrupts to capture a soundbite must only be triggered when the clock signal connected to GPIO5 is on a falling edge.
INTER-SUB INTERACTIONS	
Signal acquisition	The signal acquisition must be invoked when the interrupt is triggered.
	The processed data from the signal acquisition submodule must be numbered, named and sent back to the signal acquisition sub module.
Hardware	The clock signal and GPIO pin configuration must be configured by the raspberry pi.
	The numbered arrays of sound bites must be stored in the raspberry pi before being transmitted to the local computer via USB.

SIGNAL ACQUISITION

Brief description:

The signal acquisition subsystem encompasses using the two Raspberry Pi Zeros to read audio data from 4 Adafruit MEMS microphones, with two microphones connected to each Raspberry Pi using the I2S protocol. The subsystem also includes the transfer of the data from the Raspberry Pis via USB to a computer which processes the data.

Alternatives considered:

The possibility of connecting all four microphones to the same Raspberry Pi was considered. Research showed that this would be difficult to achieve, since the Raspberry Pi Zero only has one I2S port, and each I2S port supports at most 2 microphones. It was decided that using the two available Raspberry Pis and synchronising the data would be easier than trying to connect all four microphones to one Raspberry Pi.

Wireless transmission of data was considered. This was rejected because wireless protocols typically have lower speeds and are less reliable than USB transmission.

Processing all the data on one of the Raspberry Pis was considered. This was rejected since it is easier to use the graphical interface on a laptop than to attach one to a Raspberry Pi, and since laptops typically have better processing capabilities than Raspberry Pis. Consequently, the transfer of data from the Raspberry Pis to the laptop was added to the signal acquisition subsystem.

REQUIREMENTS	
Mic reading	Raspberry Pis must read data from the microphones at the specified sample rate
Data processing	Raspberry Pis must process the data from the microphones into a form that is easy to transmit to the laptop and is easy to use on the laptop
Data transfer to laptop	Raspberry Pis must transfer data to the laptop using connections from USB cables
SPECIFICATIONS	
Mic reading	Uses I2S data transfer protocol
	I2S sample frequency 1.4112 MHz sample frequency (for 44.1 kHz, 16-bit data for each mic)
Data processing	Raspberry Pi must take snapshots of audio data at the soft-configurable

	frequencies. Audio must be processed into numpy arrays, each of which contains a specified (soft-configurable) amount of audio data.
Data transfer to laptop	Each Raspberry Pi USB port is configured for serial communications using “USB Gadget” protocols. Serial baud rate set at 115200 bps.
INTRA-SUB INTERACTIONS	
All sub-subsystems	Data is read by the microphones, internally processed, and transferred over the serial-configured USB port.
INTER-SUB INTERACTIONS	
Hardware	Signal acquisition requires physical wiring for the connections between the Raspberry Pis and the microphones using the I2S busses. Additionally, signal acquisition requires physical USB connections from the Raspberry Pis to the laptop
Pi Synchronisation Time Delay Estimation	Signal Acquisition transfers data to the laptop, where data on the sound bite numbers is added, after which the date is transferred to the TDoA submodule
User interface	Signal Acquisition receives data about soft-configurable settings (sampling frequency, length of audio snapshots) from the user interface subsystem

TIME DIFFERENCE OF ARRIVAL (TDOA)

Brief description:

The TDOA subsystem involves calculating the time difference of arrival between two sound signals (multiple pairs) using a technique called Generalised Cross Correlation - Phase Transform or GCC-PHAT. This uses the Wiener-Khinchin theorem and a weighting function to calculate a cross-correlation and subsequently a time delay. The Phase Transform method or PHAT will be used for the weighting function as this provides high accuracy estimates even in low Signal to Noise Ratio (SNR) environments while remaining computationally .

Alternatives methods and theories considered:

- Cross correlation is the basis of GCC and involves the same techniques as GCC but it requires a large sample size to be feasible. In addition, it requires a high SNR environment, else the results will be inconclusive. The long samples will result in longer and heavy computation and it is highly unlikely that a high SNR environment is achievable in most cases.
- Time of arrival or TOA is the most basic form of time delay estimation whereby an absolute time delay is measured between each signal. This was not feasible as the signal broadcast time and signal itself would need to be known a priori. Furthermore, this would not allow for active tracking, limiting the signal to be located only when stationary and on a single occasion.
- Various other weighting functions for GCC were considered such as: M-CSP, Maximum Likelihood, Hassab and Boucher, Eckart Filter, SCOT filter, Roth Filter and Weiner filter. Between these, PHAT, SCOT and HB work best in low SNR and within these 3, PHAT is the most commonly used, and supported method.

REQUIREMENTS	
Calibration	Implement a Calibration Mode for the TDoA to reduce or eliminate constant delays caused by the surroundings.
Import	Import data from USB to Python.
Filter and Optimise	Filter sound data to reduce noise effects on the TDoA to increase accuracy.
	Reduce effects of moving sound on accuracy.
GCC-PHAT	The algorithm must calculate the TDoA between 2 sound signals.
SPECIFICATIONS	

Calibration	Go into calibration mode if UI is requested to calibrate
	Take 20 TDOA samples of a known position and compare the result to known TDOA. Repeat for 3 different positions on the grid. Determine an average time error and apply it as an additive constant to TDOA measurements.
Import	Use the built in **Serial** library from Python to read sound data from the USB port.
Filter and Optimise	Apply a low pass filter to the sound data to reduce higher frequencies and noise.
	Apply a Kalman filter to reduce noise and source velocity effects (moving/non-stationary source) [2].
GCC-PHAT	GCC will be used on sound signal pairs, which will always be the reference mic and another mic.
	Obtain cross-power spectral density function $G_{r1r2}(f)$ of a signal pair, $r1$ and $r2$, where $r1$ is the reference mic signal. [3]
	The cross-correlation is calculated using the Wiener-Khinchin theorem: $R_{r1r2}(\tau) = \int_{-\infty}^{\infty} \psi_{PHAT}(f) G_{r1r2}(f) e^{j2\pi f\tau}$ Where $\psi_{PHAT}(f)$ is the PHAT weighting function: $\psi_{PHAT}(f) = 1/\left\lvert G_{r1r2}(f)\right\rvert$ And τ is the time delay between $r1$ and $r2$. [3]
	The delay can be estimated by $D = argmax[R_{r1r2}(\tau)]$ [3].
	Signal Speed = 343m/s
INTRA-SUB INTERACTIONS	
All sub-subsystems	Algorithm and Optimizations to be implemented in the Code
INTER-SUB INTERACTIONS	
Signal Acquisition	Receive sound signal data from each mic in discrete form.

Triangulation	Produce TDOA for each mic, with respect to the reference mic.
User Interface	Receives and sends Calibration mode on or off state.

TRIANGULATION

Brief description:

The Triangulation subsystem involves using the TDOA, the known mic positions in a defined grid space and the speed of the signal to estimate the position of the signal source. The method of choice is Multilateration whereby the above parameters are used to calculate various hyperbolas. These are then solved simultaneously using Linear algebra and the solution is the best estimate of the source's location.

Alternatives methods:

- Trilateralization
- Binaural Recording
- Beamforming

The above methods were some combination of: computationally expensive, realistically impractical, lacking in functionality or unnecessarily complex. Multilateration offered the best solution for active tracking of a sound source in 2D using simple microphones.

REQUIREMENTS	
Generate Hyperbola	Use TDOA to estimate a position on the source
Estimate Points	Must be able to live track
Uncertainty	Provide an uncertainty of the estimated position
Output to UI	Combines data and [X, u(X), Y, u(Y), sample number]
SPECIFICATIONS	
Generate Hyperbola	Multilateration is used to triangulate the sound source.
	Calculate 3 hyperbolas using 4 mics: $\Delta d = c_s * (\Delta t)$ where Δd is the difference in distance between the two mic and the source, c_s is the speed of sound at 343m/s and Δt is the TDOA. $\Delta d = \sqrt{(x_2 - x)^2 - (y_2 - y)^2} - \sqrt{(x_1 - x)^2 - (y_1 - y)^2}$ Where x1 and x2 and position of the reference mic and other mic. [4][5][1]
Estimate Points	X and Y points are calculated by solving the hyperbolas using linear algebra (Matrices). There may be 1, 2, or 3 solutions.

	Midpoint theorem is used on solved solutions of hyperbolas to calculate estimated position of source.
Uncertainty	Uncertainty is calculated based on Uncertainty analysis of midpoint theorem, X position uncertainty, u[X] and Y position uncertainty, u[Y].
Output to UI	Compile uncertainty, estimated point and sample number as an array: [X, u(X), Y, u(Y), sample number] for UI
INTRA-SUB INTERACTIONS	
All sub-subsystem	Each sub-subsystem is a method that will pass its values to the next.
INTER-SUB INTERACTIONS	
TDOA	Receives TDOA and sample number which is used to estimate position.
User Interface	Sends estimated position and uncertainty to UI to be processed and displayed

USER INTERFACE

Brief description:

The user interface is the final point of connection to the user. It shows the location of the sound source within the grid in a meaningful way that is easy for the user to understand. This could be a rendered “map” of the area with a location of the sound source, or an option to export data about the position to a file. Additionally, the user interface is used to accept input for details such as when to start/stop recording, sample rates, and calibration procedures.

The user interface will be graphical, and display

Alternatives considered:

A barebones command-line interface will likely be implemented in addition to the graphical interface for testing during the design procedure. However, the higher-level requirements specify that the final interface should be graphical, thus this will be the ultimate goal for this subsystem.

REQUIREMENTS	
Real time plot	Interface must render a representation of the grid, with a clearly visible marker indicating the current position of the sound source on the grid.
	The location of the marker should be based on the best estimation of the location of the sound source and not show uncertainty, as this will be shown on the heatmap.
	A toggle-able option must allow the plot to show previous locations of the marker with different colours indicating the time since the marker was in the position.
Heatmap	Interface must render a representation of the grid with a heatmap showing the likelihood of the sound source being in specified areas based on the degree of uncertainty in the measurements
Real time plot & Heatmap	Grid size must be soft-configurable, with the default size being the size of the physical A1 grid
Output to file	Interface must allow the option to export data to a comma separated values (csv) file
	Path to output file must be configurable
Calibration procedure	Interface must show prompts on how to calibrate the system and receive necessary user input about the calibration process

Settings	Interface must include soft-configurable settings such as sampling frequency
Interface controls	Interface must have easily understandable controls
SPECIFICATIONS	
All sub-subsystems	User interface should be coded in Python using pygame
Real-time plot	Def
	Grid lines should be faint grey. Markers should be green, and marker history should be displayed with a gradient of colours from green to red to blue, with green being the most recent marker, and blue being the oldest.
Heatmap	Heatmap should display a gradient of colours from green to blue to red, with green being the area with the highest probability of containing the sound, and blue being the area with the lowest probability of containing the sound
Output to file	File must be in comma separated value (csv) format
	Each line on the file should represent data in the form: <x-position>, <x-position standard uncertainty>, <y position>, <y-position standard uncertainty>, <sample number>, <time>
	Must be included in the command-line interface
Calibration procedure	Display must show instructions about where to place the sound source, including written instructions and a marker on the real-time plot
	A button must be clearly visible that the user clicks once they have placed the sound source in the relevant position.
	Must be included in the command-line interface, with only the text instructions, and prompts to press “Enter” when the sound source has been placed in the relevant position
Settings	Must contain settings to soft configure: - Size of grid - Sampling frequency - Recording start and stop times
	Must be included in the command-line interface
Interface controls	All controls must be accessible through GUI buttons. Some controls can also be included in the command-line interface, as per their

	individual specifications.
INTRA-SUB INTERACTIONS	
Interface controls - all other sub-subsystems	Interface controls include options to switch between modes (real time plot, heatmap, calibration)
Interface controls - real time plot	Must include controls to pause live tracking and/or turn tracking history on and off
Interface controls - heatmap	Must include controls to pause live tracking
Interface controls - settings	Must include controls to edit soft-configurable settings
	Also included in command line controls
Interface controls - output to file	Must include controls to output data to file, and settings for the outputted data (number of data points, sampling frequency, file locations)
Interface controls - calibration	Must include instructions about how and where to position the sounds on the grid for calibration procedure, and buttons to allow users to feed back when they have placed the sound at the correct location
INTER-SUB INTERACTIONS	
Hardware	User interface is displayed on the computer - written in python so should be compatible cross-platform
Signal acquisition / triangulation	User interface receives data that is displayed on the interface, as per the specifications from the signal acquisition and triangulations subsystems
All other subsystems	User interface outputs soft-configurable control settings which affect the functioning of the other subsystems. Software for other subsystems must include functions which can receive soft-configurable settings that are relevant for that subsystem.

HARDWARE

Brief description:

This subsystem is made up of the various different physical components used in this project. The hardware subsystem stretches across multiple subsystems and is the basis of where all subsystems will be acting. This subsystem focuses on the specifics of configuring the physical components that are responsible for executing the various subsystems.

Alternatives considered:

Using only the two raspberry pi zeros as the processing units that are responsible for all the computations and hosting the GUI but after careful consideration, it was decided that the pi's may not have enough processing power to support real-time tracking.

REQUIREMENTS	
Power supply	The local laptop must be powered by its inbuilt battery or charger.
	Each pi must be powered via a phone charger connected to a wall socket or extension lead.
Mic and Pi Placement	Each microphone must be placed in a jig that is fixed to the grid so that the mic does not move during localization.
	Each mic must be placed on a corner of the grid.
	Each raspberry pi must be placed next to one of its connected mics.
Wiring	A local laptop must be connected to both pi's via two separate USB cables.
	There are four microphones in total and two microphones must be connected to each raspberry pi.
	The relevant GPIO pins of the two pi's must be connected using UTP wiring.
Local Laptop	The laptop must contain hardware that is able to process the data received at a fast enough rate to ensure that there is minimal delay between when the sound bite is captured and when its resulting position is displayed by the user interface.
Raspberry Pi's	The two raspberry pi zero w's must be fast enough to capture the sound bites from their connected mic's and transmit the data to the local laptop before the next sound bite is captured.

Microphones	There must be four identical microphones that use I2S as their transfer protocols.
SPECIFICATIONS	
Power supply	The laptop's battery must be able to last at least 3 hours.
	The phone charger used must be rated at 5V and 2A.
Mic and Pi Placement	The jig used to hold the mic must be made of firm cardboard and must be fastened to the grid using presstick.
Wiring	The mic closest to the raspberry pi will be connected using standard arduino jumper cables and the mic furthest away from the Pi must be connected using CAT 3 UTP wires.
	The usb cables connecting the local laptop and the raspberry pi's must be micro USB 2.0.
	The wire connecting the two raspberry pi's GPIO pins together must be CAT 3 UTP wires.
Local Laptop	Must have the following minimum specifications: - 8GB RAM - 2GHz, quad-core CPU - 2GB free space
	Must have the following software installed: - Matlab R2023a - Python3 - Unix OS - Two USB 2.0 ports
Raspberry Pis	The raspberry pi zero w's used have the following hardware specifications: - 1GHz, single-core CPU - 512MB RAM - I2S Capabilities - Micro-USB data connection - 16GB micro SD memory card
	The raspberry pi's must have the following software installed: - Raspberry Pi OS - Python3

Microphones	The microphones used must be the Adafruit I2S MEMS microphone breakout boards. They have the following specifications: - I2S Output - 1.6-3.6V input voltage - 32KHz to 64KHz sampling rate
INTRA-SUB INTERACTIONS	
Power Supply - Raspberry Pi's	The raspberry pi's are supplied with power by two phone chargers
Power Supply - Local Laptop	The laptop is powered by its battery.
Local Laptop - Raspberry Pi	The local laptop is connected to both raspberry pi's via two USB 2.0 cables across which the raspberry pi will send the local computer sound bite data and the local computer will send instructions to the raspberry pi.
Microphones - Raspberry Pi's	The microphones record sounds and transfer them to the raspberry pi's using the I2S transfer protocol.
INTER-SUB INTERACTIONS	
Signal Acquisition	The microphones capture the signals and transfer the captured signals to the raspberry pi using the I2S transfer protocol.
TDOA	The TDOA is computed using python3 on the local laptop.
User-Interface	The local laptop is where the GUI will be run and the user can interface with the various controls.
Pi Synchronisation	The raspberry pi is responsible for configuring the GPIO pins that govern the synchronisation process.

ACCEPTANCE TEST PROCEDURES

Figures of merits for validation:

Pi Synchronisation:

1. There is the same number of sound bites for all four microphones.
2. For every clock cycle there is a set of data for all microphones.
3. Clock signal is 10 Hz.

Signal Acquisition:

4. Microphones are able to read a 10 second audio clip at 44.1 kHz sample frequency, 16 bit resolution
5. 4 numpy discrete sound magnitude arrays are correctly collected.

TDOA:

6. Measured TDoA matches theoretical TDoA within 1% tolerance.
7. Estimated TDoA is still accurate under noisy conditions.

Triangulation:

8. Estimated position of hyperbolas are within 5% tolerance when source is moving at maximum speed of 0.1m/s.

User interface:

9. Graphics update correctly every 0.25 seconds.
10. User interface receives qualitative approval on the basis of useability from at least 5 different people
11. All buttons on user interface perform expected behaviour when clicked

Experiment design to test these figures of merit.

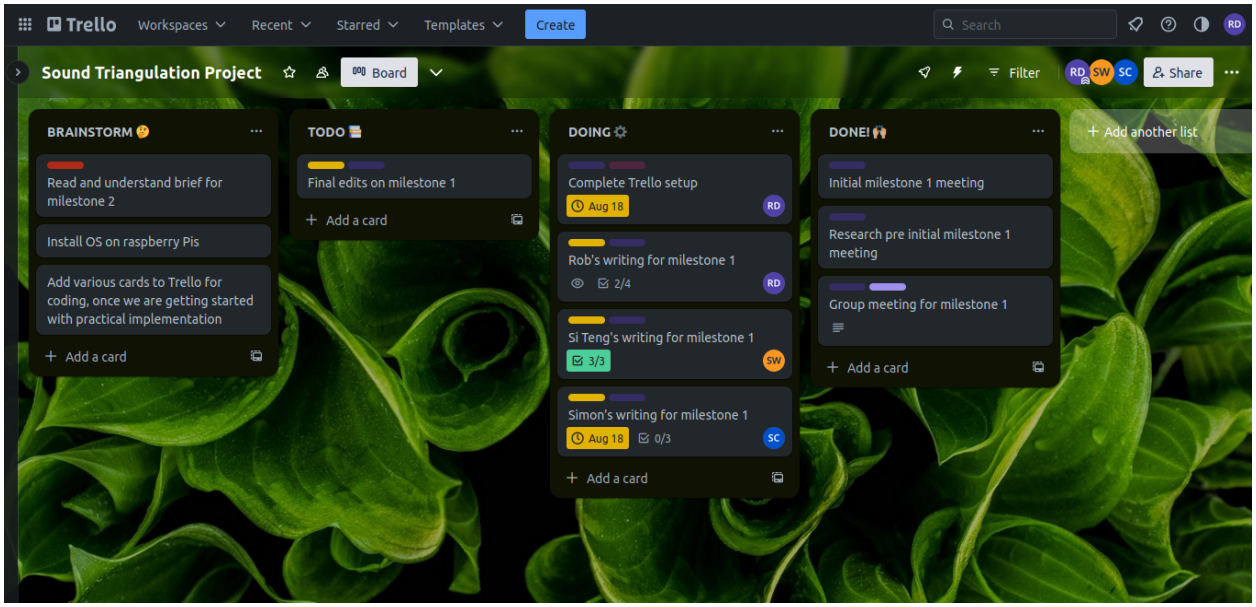
1. Generate an array of timestamp data from the two Raspberry Pis over a number of interrupt triggers and see if the master and slave Pi output the same data in the arrays. This will show if the data collection will yield the same results as they are starting at the same time.
2. Use a 1 kHz sine wave sound source and check if each mic returns similar data every clock cycle.
3. Connect the output of the clock signal to an oscilloscope and measure the frequency.
4. Record a 5 Hz signal to each mic and export the data to Python. Analyse the data for the correct number of discrete points, bit resolution and record time.
5. Play a 1 kHz signal to each mic and capture a single period. Send data and plot in Python on the main computer to check for a valid sine wave. Test is good if all 4 numpy arrays have a sine wave. Test to be conducted at the maximum and minimum distance to the mics in order to test for saturation or insufficiency.

6. Create 4 identical numpy arrays with a 1 kHz sine wave and a delay on each. Input the array into the TDOA method and check results. Then use actual recorded data from mics and check results. Test is good if TDOA is within 1% error from theoretical value.
7. Obtain recording of the lab when noisy and obtain a SNR ratio by doing a power spectrum analysis. Simulate the TDOA algorithm with a 1kHz sine wave in 4 numpy arrays with the added noise and a known time shift and check if results are within 1% tolerance.
8. Simulate a scenario of a moving signal in MATLAB and check if the algorithm is correct. Then create various TDOA samples with known points on the grid and compare results to actual position. Finally, move the sound source on known paths in the grid space and compare the estimated paths to the actual paths. If accuracy is within 5% tolerance, the test is passed.
9. Feed dummy data to the UI and test if it correctly displays everything to check performance and accuracy. The test is passed if the data is correct, and appears within expected time.
10. Ask others to try to use the UI and obtain feedback and qualitative approval.
11. Create a checklist of each functionality and test each. Check fluidity between each button/functionality to ensure no crash/freeze conditions. Test is passed if the checklist is complete.

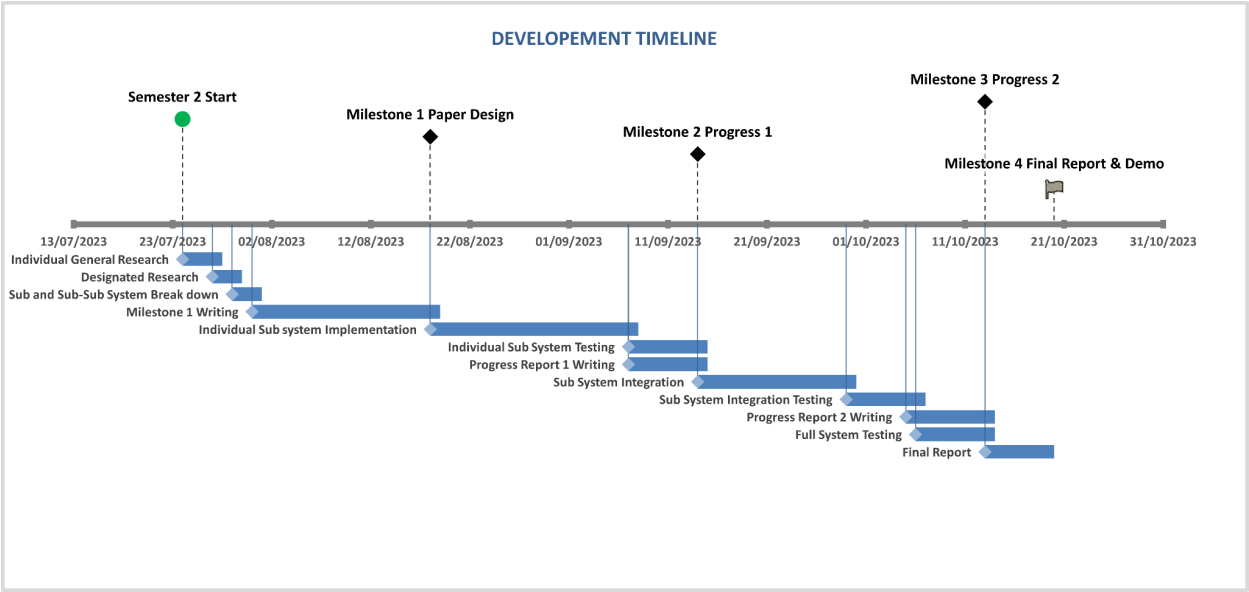
Acceptable performance definition:

The system is able to detect a moving sound source of 1kHz within a defined grid space and in a low SNR environment using 4 microphones, 2 Raspberry Pis and the main computer. The estimated position is within 5% of the actual position at a maximum linear speed of 0.1m/s and is correctly displayed in a user interface. The system test can be repeated and the hardware is reusable for multiple detections on separate occasions.

PROJECT MANAGEMENT PAGE



DEVELOPMENT TIMELINE



REFERENCES:

[1] D. Dalskov, “Locating Acoustic Sources with Multilateration - Applied to Stationary and Moving Sources,” pp. 1–6, 37–43, Jun. 2014, Accessed: Aug. 18, 2023. [Online]. Available: https://projekter.aau.dk/projekter/files/198526294/main.pdf

[2] Alex Becker (www.kalmankilter.net, “Online Kalman Filter Tutorial,” *Kalmanfilter.net*, 2016. https://www.kalmanfilter.net/default.aspx

[3] R. Boora and S. Dhull, “A Comparison of Generalized Cross-Correlation Methods for Time Delay Estimation,” The IUP Journal of Telecommunications, vol. VOL. VIII, no. NO. 4, Apr. 2016, Accessed: Aug. 18, 2023. [Online]. Available: https://www.researchgate.net/publication/340884332_A_comparision_of_Generalized_cross_correlation_methods_for_time_delay_estimation

[4] “Object Tracking Using Time Difference of Arrival (TDOA) - MATLAB & Simulink - MathWorks United Kingdom,” *uk.mathworks.com*. https://uk.mathworks.com/help/fusion/ug/object-tracking-using-time-difference-of-arrival.html (accessed Aug. 18, 2023).

[5] B. O'keefe, “ECE Senior Capstone Project 2017 Tech Notes Finding Location with Time of Arrival and Time Difference of Arrival Techniques.” Available: https://sites.tufts.edu/eeseniordesignhandbook/files/2017/05/FireBrick_OKeefe_F1.pdf