Open Acoustic Devices

Using AudioMoth GPS Sync to Make Synchronised Recordings

theteam@openacousticdevices.info

4th May 2025

The standard AudioMoth firmware supports the use of a connected GPS receiver to update its real-time clock, and to measure the precise sample rate, before each recording period. This allows events within a recording to be assigned a timestamp that is accurate to within a few milliseconds with little increase in overall energy consumption. In many applications, this level of time synchronisation is sufficient. However, sometimes more precise time synchronisation is required. For example, when the source of acoustic events must be localised to high accuracy. In these settings, AudioMoth GPS Sync can be used. This solution consists of alternative firmware that powers the GPS continuously whilst audio recordings are being made. This increases the energy consumption of the recording process, but also allows the AudioMoth to continuously track and record the arrival of pulse-per-second signals from the GPS module. An associated desktop app then performs post-processing of the recorded data to generate synchronised WAV files that are aligned with GPS time to better than 1 microsecond.

1   AudioMoth GPS Receivers

The easiest way to connect a GPS module to AudioMoth 1.1 & 1.2 is to use the AudioMoth GPS Hat (see Figure 2) or the AudioMoth GPS Board (see Figure 1) if you are using AudioMoth Dev. See the 'Using the AudioMoth GPS Board and Hat' application note for further details.[1]

The AudioMoth GPS receivers can use either the PA1616S[2] module that uses just GPS satellites to calculate position fixes, or the PA1616D[3] module that by default uses both GPS and GLONASS satellites. The PA1616D module uses more energy whilst tracking GPS satellites (typically 29mA compared to 20mA for the PA1616S module), but is more robust to poor reception
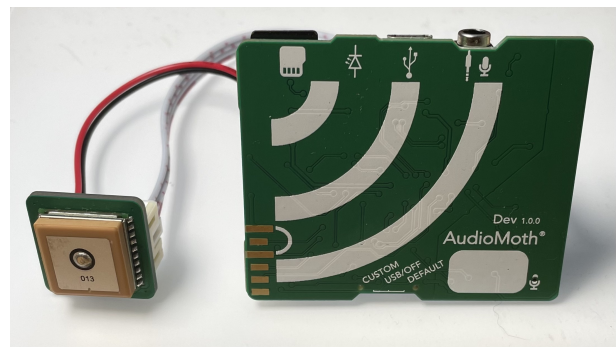

Figure 1: *AudioMoth GPS Board connected to an AudioMoth Dev board with standard JST PH jumper cables.*

Figure 2: *AudioMoth GPS Hat soldered to an AudioMoth 1.2.0 (in this case, with a 3.5mm socket).*

due to the larger number of satellites available and the wider range of frequencies at which they operate.[4]

2   AudioMoth GPS Sync

AudioMoth GPS Sync comprises two components: (i) an alternative device firmware that continuously powers the GPS module whilst making audio recordings in order to track and record the arrival of pulse-per-

---

[1] https://github.com/OpenAcousticDevices/Application-Notes/tree/master/Using_the_AudioMoth_GPS_Board_and_Hat

[2] https://www.cdtop-tech.com/products/pa1616s

[3] https://www.cdtop-tech.com/products/pa1616d

[4]Other GPS modules can be used as long as they provide standard NMEA RMC messages and provide a pulse-per-second output.

```
{
    gain:  <0, 1, 2, 3 or 4>,
    sampleRate:  <8000, 16000, 32000, 48000, 96000, 125000 or 192000>,
    enableDailyFolders:  <0 or 1>,
    enableFilenameWithDeviceID: <0 or 1>,
    enableLowGainRange:  <0 or 1>,
    enableMagneticSwitch:  <0 or 1>,
    disable48HzDCBlockingFilter:  <0 or 1>,
    enableAlternativeBatteryVoltageRange:  <0 or 1>,
    initialFixDuration:  <1 to 3600>,
    intervalToNextFixAttempt:  <1 to 3600>,
    recordingFixDuration:  <1 to 3600>,
    sleepRecordCycle:  {
        sleepDuration:  <1 to 3600>,       # sleepDuration >= recordingFixDuration
        recordDuration:  <1 to 3600>,
    },
    recordingPeriods:  [
        { startTime:  "hh:mm", endTime:  "hh:mm" },
        { startTime:  "hh:mm", endTime:  "hh:mm" }
    ],
    sunRecording:  {
        sunRecordingMode:  <0, 1, 2, 3 or 4>,
        sunRecordingEvent:  <0, 1, 2 or 3>,
        sunRoundingMinutes:  <0 to 60>,
        sunriseInterval:  {beforeMinutes:  <0 to 360>, afterMinutes:  <0 to 360>},
        sunsetInterval:  {beforeMinutes:  <0 to 360>, afterMinutes:  <0 to 360>}
    },
    firstRecordingDate:  "dd/mm/yyyy",
    lastRecordingDate:  "dd/mm/yyyy"
}
```

Figure 3: *JSON format of the* `SETTINGS.TXT` *file.*

second signals, and (ii) a desktop app that takes this data and generates WAV files that are accurately synchronised to GPS time.

2.1 Alternative Firmware

The AudioMoth-GPS-Sync firmware can be installed on any AudioMoth device (including MicroMoth and AudioMoth Dev variants) by downloading it from the AudioMoth website[5], and using the 'Use Local File' option within the AudioMoth Flash App[6] to update the device firmware.

2.1.1 Specifying Recording Settings

Rather than using the AudioMoth Configuration App for specifying recording settings, the AudioMoth-GPS-Sync firmware reads its setting from a `SETTINGS.TXT` file on the SD card. This allows it to use a wider range of settings than the standard firmware, including options to synchronise recording periods with the local time of sunrise and sunset.

The `SETTINGS.TXT` file uses a JSON format, which is shown in Figure 3. The ordering of elements is fixed but the blue elements are optional and can be omitted. Any text may be included before the settings object, as long as it does not use a '{' character. Similarly, any text can follow the settings object. AudioMoth will ignore both. Comments within the settings are denoted with a hash ('#') symbol and cause the remainder of the line to be ignored.

Fixed recording periods can be defined within the `recordingPeriods` list. Each recording period is defined by its start and end time. These recording periods must be non-overlapping and ordered by their start time. Recording periods may span midnight (e.g. starting at `20:00` and ending at `04:00`), and midnight can be denoted by `24:00`. A maximum of ten individual recording periods may be specified and all times and dates are UTC.

The `recordingPeriods` list can be replaced with a `sunRecording` block to enable recording modes associated with the calculated time of sunrise and sunset. There are five possible recording modes which are set by the `sunRecordingMode` parameter:

- Mode (0) makes one recording around sunrise; specifically, from `beforeMinutes` before sunrise

---

[5]https://www.openacousticdevices.info/gps-sync
[6]https://www.openacousticdevices.info/applications

```
1  {
2      gain: 2,
3      sampleRate: 48000,
4      enableDailyFolders: 0,
5      enableLowGainRange: 0,
6      enableMagneticSwitch: 0,
7      disable48HzDCBlockingFilter: 0,
8      enableAlternativeBatteryVoltageRange: 0,
9      initialFixDuration: 120,
10     intervalToNextFixAttempt: 60,
11     recordingFixDuration: 60,
12     recordingPeriods: [
13         { startTime: "04:00", endTime: "08:00" },
14         { startTime: "20:00", endTime: "24:00" }
15     ],
16     firstRecordingDate: "01/02/2023",
17     lastRecordingDate: "28/02/2023"
18  }
```

```
1  {
2      gain: 2,
3      sampleRate: 48000,
4      enableDailyFolders: 0,
5      enableLowGainRange: 0,
6      enableMagneticSwitch: 0,
7      disable48HzDCBlockingFilter: 0,
8      enableAlternativeBatteryVoltageRange: 0,
9      initialFixDuration: 120,
10     intervalToNextFixAttempt: 60,
11     recordingFixDuration: 60,
12     sunRecording: {
13         sunRecordingMode: 2,
14         sunRecordingEvent: 1,
15         sunRoundingMinutes: 5,
16         sunriseInterval: {beforeMinutes: 30, afterMinutes: 30},
17         sunsetInterval: {beforeMinutes: 30, afterMinutes: 30}
18     }
19  }
```

Figure 4: *Two example* SETTINGS.TXT *files.*

to afterMinutes after sunrise.

- Mode (1) makes one recording around sunset; specifically, from beforeMinutes before sunset to afterMinutes after sunset.
- Mode (2) makes two separate recordings around both sunrise and sunset, as per modes (0) and (1).
- Mode (3) records during the night; specifically, from beforeMinutes before sunset to afterMinutes after sunrise.
- Mode (4) records during the day; specifically, from beforeMinutes before sunrise to afterMinutes after sunset.

In some recording modes, either sunriseInterval or sunsetInterval may not be used, and in others either beforeMinutes or afterMinutes may not be used. However, in all cases, the unused values must be present in the file and can be set to zero, or left at some default value.

Furthermore, from AudioMoth-GPS-Sync 1.2.0 onwards, it is possible to change the reference time from sunrise and sunset, to civil, nautical or astronomical dawn and dusk using the optional sunRecordingEvent parameter:

- Event (0) uses sunrise and sunset where the sun's upper limb (edge) disappears below the horizon.
- Event (1) uses civil dawn and dusk where the sun's center is 6° below the horizon.
- Event (2) uses nautical dawn and dusk where the sun's center is 12° below the horizon.
- Event (3) uses astronomical dawn and dusk where the sun's center is 18° below the horizon.

Whether using fixed recording periods or calculated sunrise and sunset times, from AudioMoth-GPS-Sync 1.1.0 onwards, an optional sleep/record cycle can also be included. In this case, it is important to ensure that the sleep duration is greater than the recording fix duration to allow sufficient time for the GPS module to acquire a fix before the recording starts.

The firstRecordingDate option and the lastRecordingDate option work as in the standard firmware and can be omitted.

Finally, from AudioMoth-GPS-Sync 1.2.0 onwards, the optional enableFilenameWithDeviceID parameter allows the use the device ID as a prefix in the WAV file name.

The settings are read from the SD card whenever the AudioMoth switch is moved from USB/OFF to DEFAULT or CUSTOM. If there is a problem with the SETTINGS.TXT file, both LED will flash twice to indicate an error. The AudioMoth will write a LOG.TXT file describing the problem.

As with the standard firmware, switching to DEFAULT will immediately start a continuous recording, and switching to CUSTOM will use the fixed recording periods, or the sun recording mode, depending on the contents of the SETTINGS.TXT file.

When first switched to CUSTOM the AudioMoth will check if its clock has been set. If it has not, it will immediately power up the GPS and try to acquire a fix. If no fix is achieved within initialFixDuration seconds, it will power off for intervalToNextFixAttempt seconds before trying again. Once the time has been set, the AudioMoth will sleep until the scheduled recording period, waking up recordingFixDuration seconds before the scheduled time to acquire a GPS fix.

Figure 4 shows two examples SETTINGS.TXT files: one using fixed recording periods with first and last recording dates, and one using sun recording mode 2 to make one-hour recordings centred around civil dawn and dusk. In this case, the times of dawn and dusk will be rounded to the nearest five minutes, typically resulting in recording start times that change by either zero or five minutes between consecutive days to keep track of the changing time of dawn and dusk.

2.1.2 Output Files

The AudioMoth-GPS-Sync firmware generates some additional files when first switched to DEFAULT or CUSTOM mode, and when recording. When first switched to DEFAULT or CUSTOM mode, the AudioMoth will read the SETTINGS.TXT file and generate the DEVICE.TXT and LOG.TXT files. The DEVICE.TXT file describes the device serial number and the version of installed firmware. The LOG.TXT file contains a detailed description of the AudioMoth activity including any problems encountered while reading the SETTINGS.TXT file.

Figure 5 shows the contents of the LOG.TXT file after a deployment to make a single one-hour recording. The contents show the AudioMoth reporting on it initially being switched on, successfully parsing the SETTINGS.TXT file, setting the time from the GPS, and then making a recording. The full contents of the SD card after deployment is shown in Figure 6.

When a recording is made, the firmware generates a regular un-synchronised WAV file; in this case, the 20221211_210000.WAV file. In addition, it writes a CSV file with additional information about each pulse-per-second event; in this case, the 20221211_210000.CSV file. This CSV file contains columns recording the total number of microphone samples captured between each pulse-per-second event, along with the value of the timer used to trigger microphone sample acquisition and conversion at the moment that the pulse-per-second event occurred. This data, along with the un-synchronised WAV file, is used to generate synchronised WAV files that are precisely aligned with GPS time.

2.1.3 LED Flash Patterns

While the AudioMoth is acquiring the initial and pre-recording GPS fix, it will show a constant green LED with a flashing red LED. The flash pattern matches that of the standard AudioMoth firmware with a single flash every second whilst waiting for GPS messages, two flashes per second whilst receiving GPS messages and waiting for pulse-per-second events, and three flashes per second whilst it waits for an appropriate number of interleaved GPS fix messages and pulse-per-second events prior to setting the internal time.

Once the recording has started, the red LED will flash during writes to the SD card as per the standard AudioMoth firmware. In addition, the green LED will change state on the arrival of each pulse-per-second event. The green LED may stay on or off for periods longer than one second if the GPS reception is poor and the GPS module stops emitting pulse-per-second events. The recordings will continue in these cases and synchronised recordings can still be generated.

2.2 Desktop Application

The AudioMoth GPS Sync App is used to generate synchronised WAV files from the raw WAV and CSV files generated by the AudioMoth-Sync-GPS firmware (see Figure 7). The app uses the same options as the AudioMoth Configuration App to select source files or folders, add a prefix to the output file names, and export to a specified directory. All generated file names are automatically appended with _SYNC such that the file generated from those shown in Figure 6 will be 20221211_210000_SYNC.WAV. Appendix A described in detail how the data within the CSV file is used to generate the synchronised WAV file.

In cases where the GPS signal reception is poor, the GPS module may occasionally fail to generate pulse-per-second events. By default, the AudioMoth GPS Sync App will detect these events and generate an error. However, in most cases, these periods of missing events are very short — typically ranging from just a few seconds to a few minutes — and the synchronised
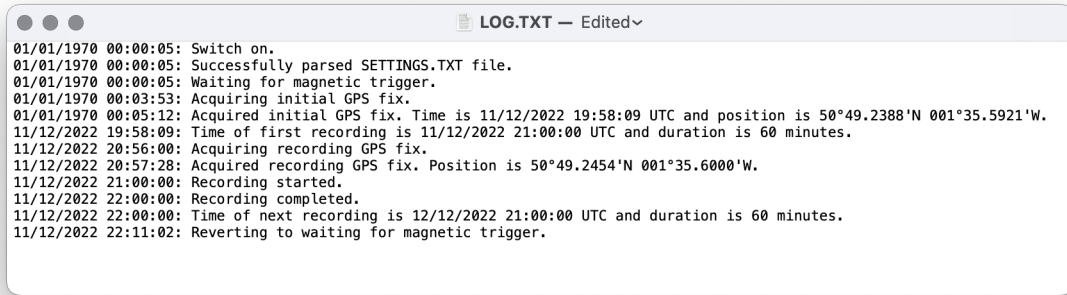
```
● ● ●                    📄 LOG.TXT — Edited∨
01/01/1970 00:00:05: Switch on.
01/01/1970 00:00:05: Successfully parsed SETTINGS.TXT file.
01/01/1970 00:00:05: Waiting for magnetic trigger.
01/01/1970 00:03:53: Acquiring initial GPS fix.
01/01/1970 00:05:12: Acquired initial GPS fix. Time is 11/12/2022 19:58:09 UTC and position is 50°49.2388'N 001°35.5921'W.
11/12/2022 19:58:09: Time of first recording is 11/12/2022 21:00:00 UTC and duration is 60 minutes.
11/12/2022 20:56:00: Acquiring recording GPS fix.
11/12/2022 20:57:28: Acquired recording GPS fix. Position is 50°49.2454'N 001°35.6000'W.
11/12/2022 21:00:00: Recording started.
11/12/2022 22:00:00: Recording completed.
11/12/2022 22:00:00: Time of next recording is 12/12/2022 21:00:00 UTC and duration is 60 minutes.
11/12/2022 22:11:02: Reverting to waiting for magnetic trigger.
```

Figure 5: *Contents of the* LOG.TXT *file written by the AudioMoth-GPS-Sync firmware.*
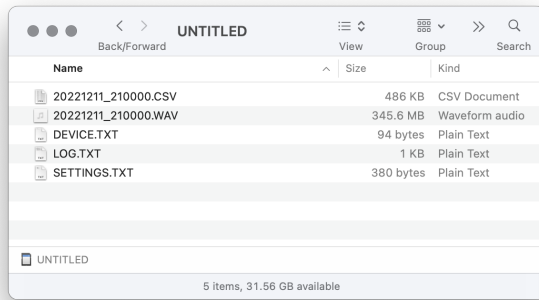
Figure 6: *Files written by the AudioMoth-GPS-Sync firmware.*
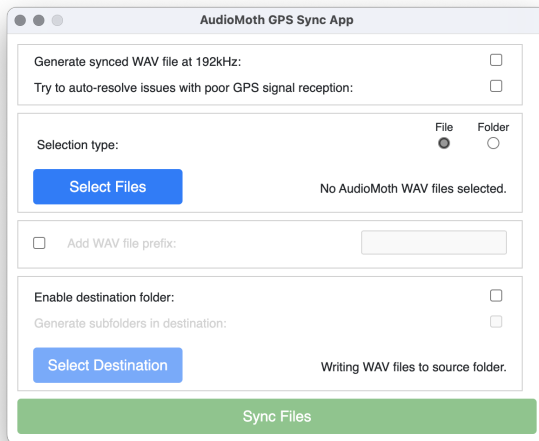
Figure 7: *AudioMoth GPS Sync App showing settings and file selection controls.*

WAV file can be generated despite them. An option allows the app to attempt to automatically resolve these periods of missing GPS data without generating errors.

In addition, by default the generated synchronised WAV file will have the same sample rate as the raw un-synchronised WAV file. However, an option allows all files to be generated at the maximum possible acquisition rate of 192 kHz. This allows files recorded
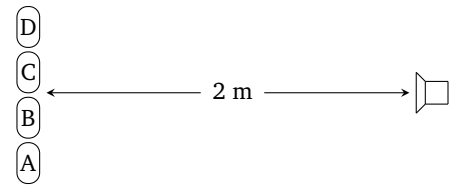

Figure 8: *The arrangement of four AudioMoth and sound source during the desktop experiment.*

at different sample rates to be easily correlated and allows more precise correlation of recordings made at low sample rates when performing discrete sample-by-sample correlation.

3   Sound Source Localisation

The ability to generate synchronised WAV files aligned with GPS times allows the source of sounds within the recordings to be localised. The accuracy of doing so depends upon the frequency and nature of the source, the sampling rate and quality of the recordings, and the degree of synchronisation between the recordings.

3.1   Desktop Experiment

To evaluate the absolute accuracy of time synchronisation between recordings, four AudioMoth were deployed a fixed distance from a speaker (see Figure 8) in a lab environment with minimal background noise. Each recorded at a sample rate of 48kHz and were subject to a sequence of short chirps, played at one-second intervals, each consisting of a frequency sweep from 8 kHz to 2 kHz over 20 ms. The synchronised WAV files were generated at 192 kHz.

For each chirp, the time offset between each pair of AudioMoth recordings (for example, t_{AB}, the time offset between AudioMoth A and B,) was found by taking a short clip around each chirp, r_A and r_B, and calculating the sample offset, j, that maximised the

per-sample correlation, such that:

$$t_{AB} = \frac{1}{f_s} \arg\max_j \mathrm{Corr}(r_A, r_B)_j$$

where f_s is the sample rate of the synchronised WAV file. For clips of N samples, the correlation function is given by:

$$\mathrm{Corr}(r_A, r_B)_j = \sum_{k=0}^{N-1} r_A^{k+j} r_B^k$$

This correlation can be calculated efficiently using fast Fourier transforms. By the discrete correlation theorem, the correlation is given by:

$$\mathrm{Corr}(r_A, r_B)_j \longleftrightarrow R_A R_B^*$$

where $\longleftrightarrow$ denotes Fourier transform pairs such that $r_A \longleftrightarrow R_A$ and $r_B \longleftrightarrow R_B$, and the asterisk denotes a complex conjugate product. This means that the offset between any two clips can be found by multiplying the FFT of one clip by the complex conjugate of the FFT of the other, taking the inverse FFT of the result, and then finding the index at which this function has a maximum value. The discrete correlation theorem only holds for periodic function. Thus, one-second clips were used, and all sample values other than the half-second interval immediately around the chirp were set to zero.

The six pair-wise time offsets were then used to estimate the longitudinal offset in the position of each AudioMoth, using the position of AudioMoth A as a reference. The time offsets define a set six simultaneous equations, such that $AX = B$, where:

$$A = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 1 \end{bmatrix} \quad X = \begin{bmatrix} x_{AB} \\ x_{AC} \\ x_{AD} \end{bmatrix} \quad B = \begin{bmatrix} v.t_{AB} \\ v.t_{AC} \\ v.t_{AD} \\ v.t_{BC} \\ v.t_{BD} \\ v.t_{CD} \end{bmatrix}$$

and v is the speed of sound and x_{AB} is the longitudinal offset between AudioMoth A and B. The matrix of offsets can then be found by least-squares fitting by evaluating $X = \left(A^T A\right)^{-1} A^T B$.

The average displacement of AudioMoth B, C and D over 60 chirps, relative to AudioMoth A, was found to be 0.2, -1.0 and 1.4 mm respectively. This was within the accuracy of the experimental setup. The standard deviation over these sixty estimates was ± 0.5 mm, which corresponds to a time offset of just ± 1.5 μs, demonstrating the ability to correlate recordings at a fraction of the original inter-sample spacing.

3.2 Field Experiment

The above experiment was repeated with the four AudioMoth deployed outside around a central sound
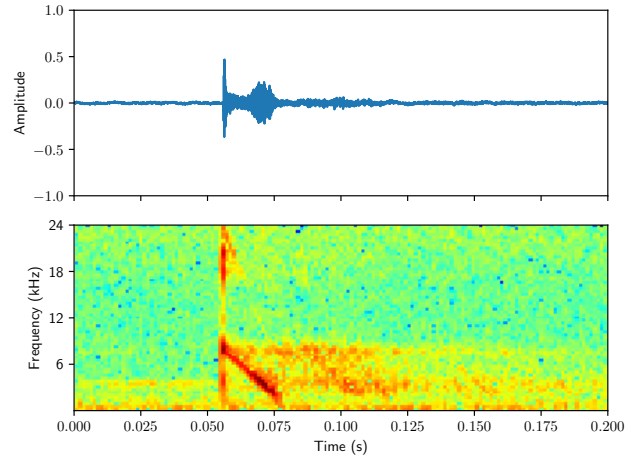

Figure 9: *Example chirp recorded by one of the four AudioMoth sampling at 48kHz.*

source (see Figure 10). In this case, the environment was less controlled with wind and bird noise in the background (see Figure 9).
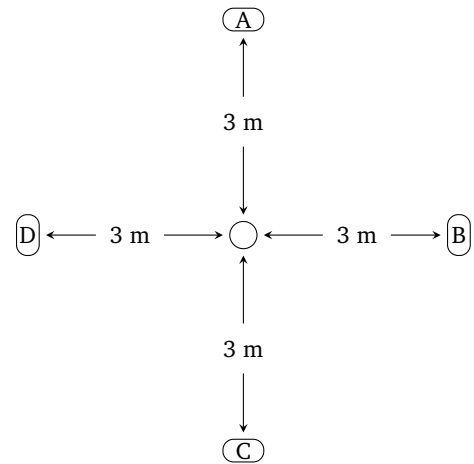

Figure 10: *The arrangement of four AudioMoth and sound source during field the experiment.*

As before, the time offset between each pair of AudioMoth recordings was found by correlating one-second intervals around each chirp. Each offset now defines a hyperbolic curve on which the sound source is located. In this case, there is no closed-form expression for the estimated position of the sound source. However, a least-squares solution to the intersection of these curves can be found using a standard Python optimisation library.[7]

Figure 11 shows the estimated source location for two groups of sixty chirps: one at the start of the recording and another an hour later. The centre of the plot is the median of both groups, and a circle with a di-

---

[7]See the example `position_from_time_differences.py` code that accompanies this application note. The code is based on the approach in `https://github.com/jurasofish/multilateration`.
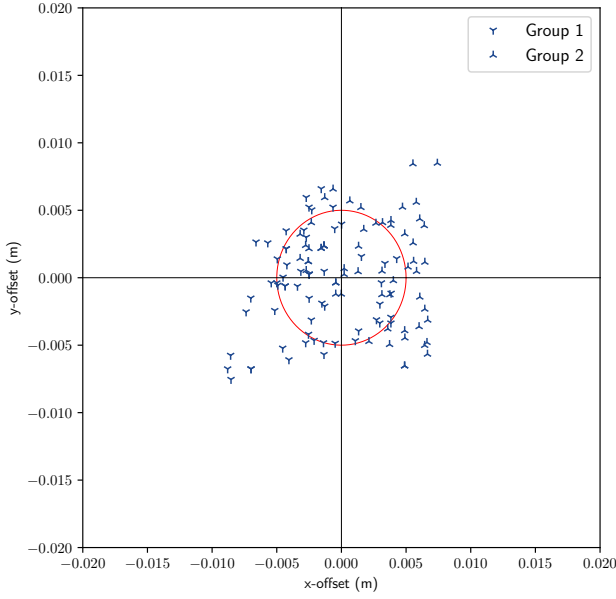
Figure 11: *Estimated source location for two groups of sixty chirps: one at the start of the recording (Group 1) and another an hour later (Group 2).*

ameter of 1 cm is shown for reference. The observed standard deviation of ± 4 mm corresponds to a time offset that is again less than the original inter-sample spacing, and there is no significant change in the mean position across the two groups over the course of the hour-long recording.

4 Conclusions

The AudioMoth GPS Sync solution, consisting of an alternative device firmware and associated desktop app, described in this application note allows AudioMoth to make recordings that are accurately aligned with GPS time. We expect AudioMoth GPS Sync to allow many novel applications involving large numbers of synchronised AudioMoth devices, and we plan to develop further software tools to assist in the visualisation and localisation of sound sources within such deployments.

Appendix

A Generating the Synchronised WAV File

As described earlier, the AudioMoth-GPS-Sync firmware writes an additional CSV file whilst recording. This file contains a record of data received from the GPS module between each pulse-per-second event. In addition, it records the total number of microphone samples captured between each pulse-per-second
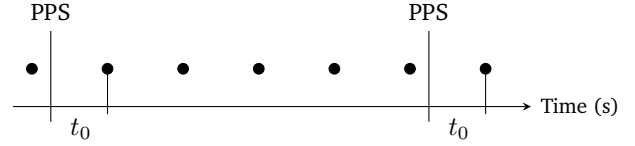

Figure 12: *Diagram showing the timing of microphone samples between pulse-per-second events.*

event, along with the value of the timer used to trigger microphone sample acquisition and conversion at the moment the pulse-per-second event occurred. The former is captured in software by counting the microphone sample interrupts in which the samples are written to the output buffer. The latter is captured by hardware at the precise moment that the pulse-per-second output from the GPS module transitions from low to high, and then read within a subsequent interrupt.

The timer value at each pulse-per-second event allows the precise timing of all audio samples in the original unsynchronised WAV file to be accurately calculated. In more detail, the microphone sample acquisition and conversion starts when the TIMER2 timer overflows and resets its counter value to 0. The period of this timer in processor cycles, N_s, is given by:

$$N_s = \frac{F_{clk}}{F_s}$$

where F_{clk} is the processor clock rate of 48MHz and F_s is the requested sample rate. Note that N_s must always be an integer value to yield an accurate sample rate.

The completion of the microphone sample acquisition and conversion process triggers a microphone sample interrupt which occurs N_{int} clock cycles after the TIMER2 timer overflows. This duration is determined by a number of analogue-to-digital converter (ADC) settings and is given by:

$$N_{int} = 2 + N_{div}\left[2 + N_{ovr}\left[N_{acq} + N_{con}\right]\right]$$

where N_{div} is the ADC clock divider (set to 4 in the firmware so that the ADC clock speed limit of 13MHz is not exceeded), N_{acq} is the sample acquisition time (set to 16 ADC cycles in the firmware), N_{con} is the sample conversion time (fixed to 12 ADC cycles by the ADC hardware), and N_{ovr} is the hardware oversample rate. This rate is chosen to maximise the recording quality, given the need to achieve the requested sample rate, and is set to 32, 16, 8, 8, 4, 2 and 2, for sample rates 8, 16, 32, 48, 96, 125 and 192 kHz, respectively.

Thus, if the TIMER2 counter has a value of N_{pps} at the pulse-per-second event, and the pulse-per-second interrupt has occurred before the microphone sample interrupt, such that $N_{pps} \leq N_{int}$, then the microphone sample interrupt that follows the pulse-per-second

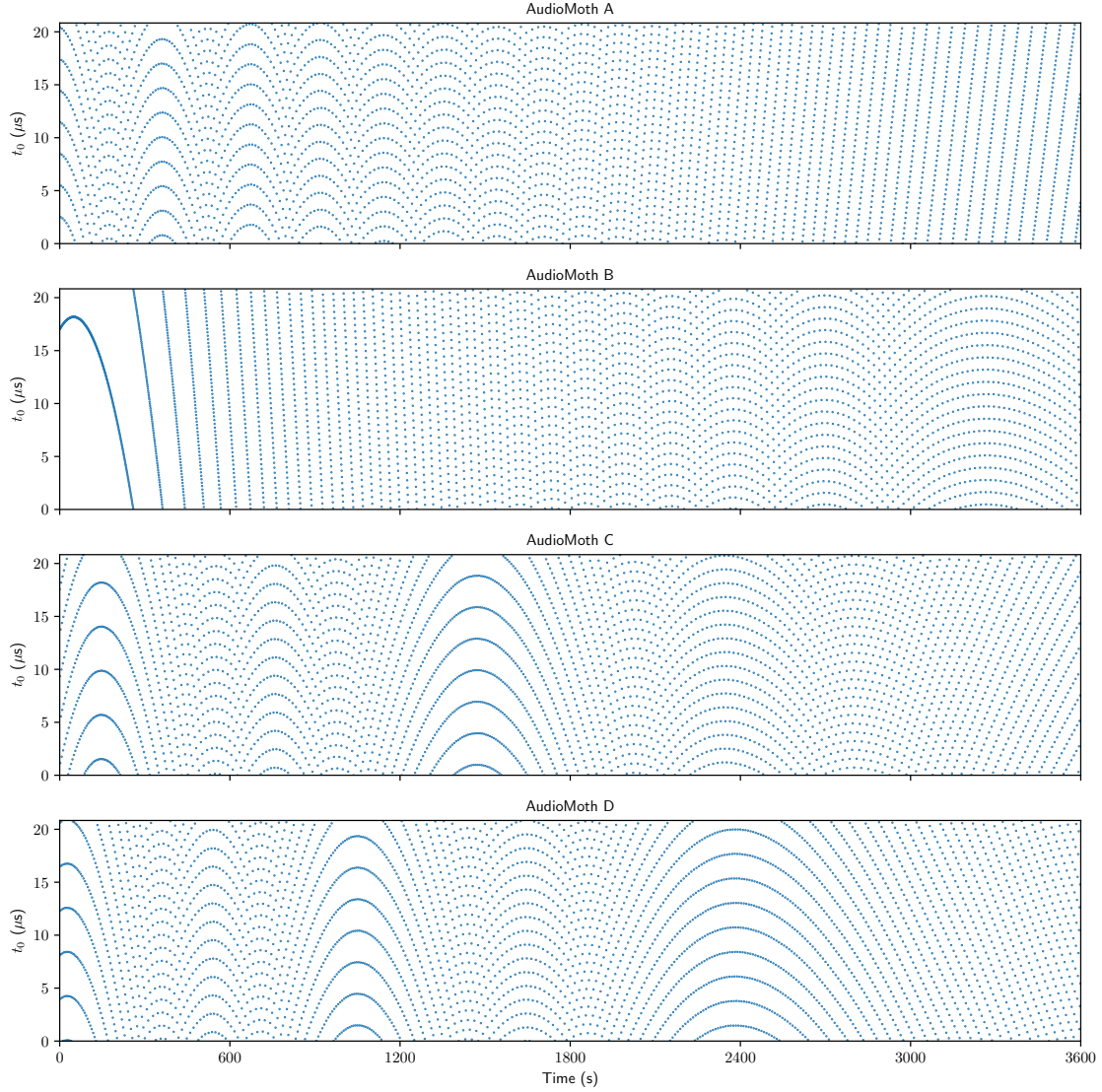**Figure 13:** *Measurements of t_0 for four AudioMoth devices at each pulse-per-second event over a one-hour recording.*

event, will occur t_0 seconds later, such that:

$$t_0 = \frac{N_{\text{int}} - N_{\text{pps}}}{F_{\text{clk}}}$$

Alternatively, if the pulse-per-second interrupt has occurred in the interval between the microphone sample interrupt and the overflow of TIMER2 to start the next acquisition cycle, such that $N_{\text{pps}} > N_{\text{int}}$, then t_0 is given by:

$$t_0 = \frac{N_{\text{s}} + N_{\text{int}} - N_{\text{pps}}}{F_{\text{clk}}}$$

Figure 13 shows the result of plotting t_0 for each pulse-per-second event of a one-hour recording in a setting where four AudioMoth devices have been removed from a warm interior to a cold exterior. For example, the value of t_0 at the first ten pulse-per-second events experienced by AudioMoth D are 8.1, 20.6, 12.3, 4.0, 16.5, 8.2, 20.7, 12.4, 4.1, and 16.6 μs. The plots show the value of t_0 varying smoothly as the samples move

in and out of phase with the pulse-per-second signal as the actual sample rate of the AudioMoth changes during the recording.

Using the count of the number of microphone sample interrupts that have occurred between each pulse-per-second event, the time interval between each sample at the requested sample rate, and the calculated value of t_0, the precise timing of each sample in the unsynchronised WAV file can be calculated (see Figure 12).

In practice, this is complicated by two additional factors. First, since the count of microphone samples is performed in software, and the capture of the TIMER2 counter value is performed by hardware, it is possible that a microphone sample can be attributed to the wrong side of the pulse-per-second event if the interrupts occur close enough together in time. Second, at the highest sample rate of 192 kHz, handling the pulse-per-second interrupt can cause a delay in handling the microphone sample interrupt that follows it. This can
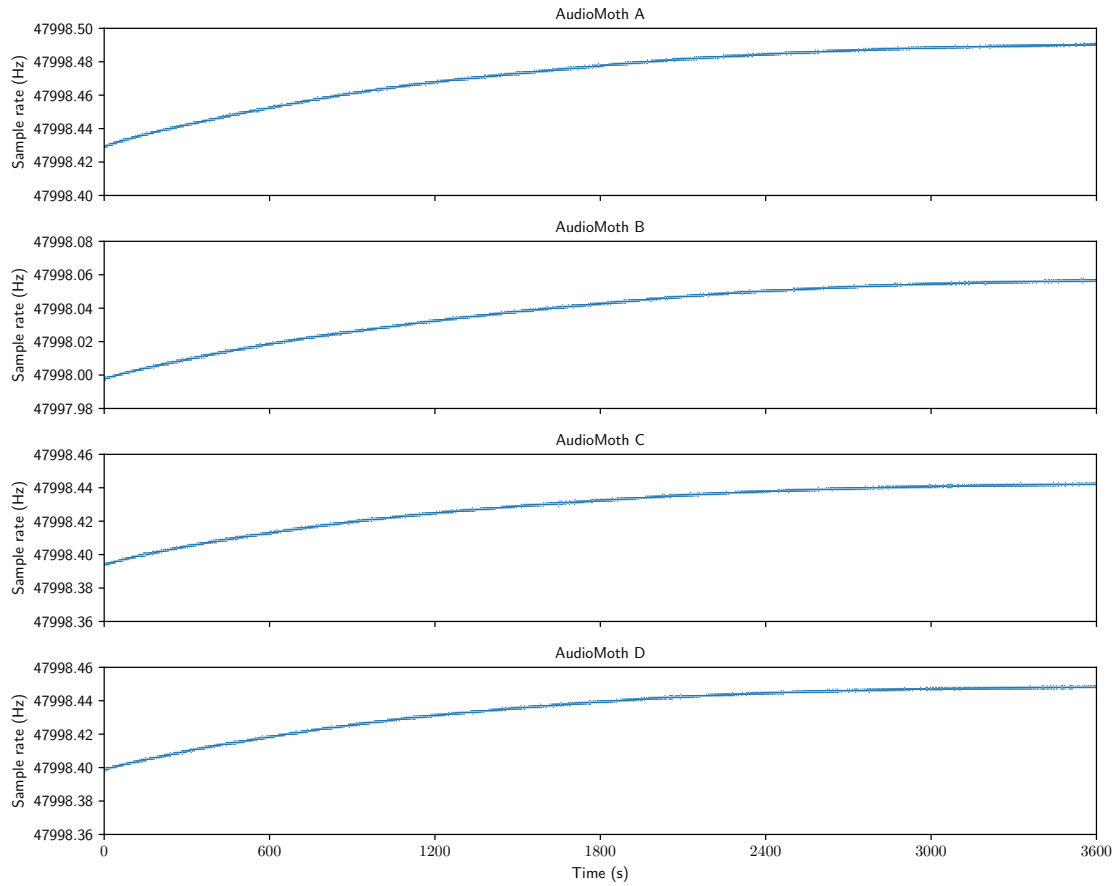
Figure 14: *Calculation of actual sample rate within each one-second interval between pulse-per-second events, for four AudioMoth devices, over a one-hour recording.*
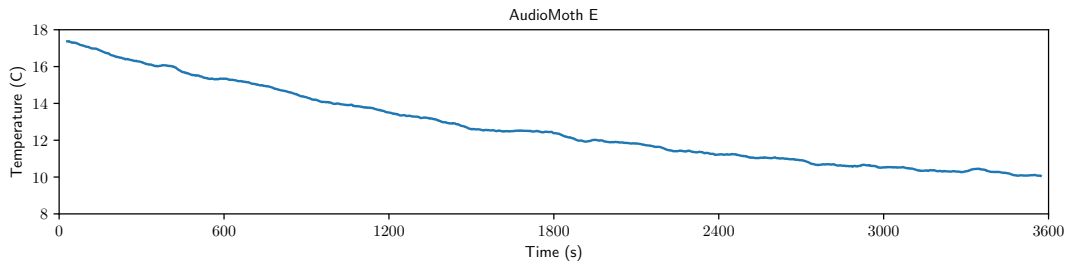

Figure 15: *Measurement of internal temperature of a fifth AudioMoth over a one-hour recording after being removed from a warm interior to a cold exterior.*

cause a sample to be missed.

Both these events can be detected by calculating the precise sample rate within each one-second interval between pulse-per-second events. This sample rate can be calculated to a precision of F_s/F_{clk} hertz. For example, the calculation is accurate to 0.001Hz when the sample rate is 48000Hz. Incorrectly attributed samples appear as a deviation in the interval sample rate of approximately 1Hz and can be easily fixed by attributing the sample to the appropriate interval, correcting t_0 in the case that a sample has been missed, and then recalculating the sample rate.

Figure 14 shows the calculated sample rate for all 3600 one-second intervals within the one-hour recording shown earlier. The difference between the actual sample rates of the four devices, and the nominal rate of 48kHz, results in time offsets of 114.6, 147.2, 117.9 and 117.4ms respectively over the course of the one-hour recording. Using an average sample rate over this period would still result in an offset of several milliseconds. By contrast, in this approach, the time of every sample in the original unsynchronised WAV file can be calculated to an accuracy of better than a microsecond.

Figure 15 shows the internal temperature (plotted with a 60-second moving average) of a fifth AudioMoth deployed with custom firmware that logs the processor

temperature every second. Note that the actual sample rate of the four AudioMoth varies with the temperature of the device due to the characteristics of the high-frequency crystal oscillator (HFXO).

Given the precise timing of all the samples within the original unsynchronised WAV file, it is then finally possible to resample the file with the intended sample rate, interpolating between the samples in the unsynchronised WAV file as necessary, to generate the synchronised WAV file. In doing so, a time offset is applied to account for the fact that the microphone sample interrupt occurs at the end of the acquisition and conversion period, N_{int}/F_{clk}, but should more accurately reflect the value at the mid-point of this period. Resampling allows us to generate the synchronised WAV file at the same sample rate as the original unsynchronised WAV, or a higher sample rate if necessary.

The software used by the AudioMoth GPS Sync App to perform this processing is included in the Node.js AudioMoth-Utils repository.[8] It is also included within the audiomoth-utils npm package and can be used within other applications or command line tools.[9]

---

[8] https://github.com/OpenAcousticDevices/AudioMoth-Utils

[9] https://www.npmjs.com/package/audiomoth-utils