STUDY OF ADAPTIVE SAMPLING, ADAPTIVE RESOLUTION AND INEXACT DESIGN APPLIED TO SENSOR NETWORKS

A THESIS

Submitted in partial fulfillment of the requirements
for the award of the degree of

BACHELOR OF TECHNOLOGY *in*
ELECTRICAL ENGINEERING

&

MASTER OF TECHNOLOGY *in*
MICROELECTRONICS AND VLSI DESIGN

By

SAKETH BSV
EE10B071

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, MADRAS

MAY 2015

ACKNOWLEDGMENTS

I would like to express my sincere thanks to my guide **Dr. G Venkatesh** for his motivating guidance with suggestions and discussions at various stages throughout my project work. I am also quite grateful to my friends for helping me refine my ideas and family for supporting me throughout.

Saketh BSV

THESIS CERTIFICATE

This is to certify that the project report entitled **"*STUDY OF ADAPTIVE SAMPLING, ADAPTIVE RESOLUTION AND INEXACT DESIGN APPLIED TO SENSOR NETWORKS*"** submitted by **Saketh BSV** (EE10B071) to the Indian Institute of Technology Madras for the award of the degree of **Bachelor of Technology** and **Master of Technology in Electrical Engineering is** a bonafide record of research work carried out by him under my supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. G Venkatesh

Professor (Guide)

Department of Electrical Engineering

Indian Institute of Technology Madras

Chennai – 600 036

ABSTRACT

Any aspect of the world can usually be seen in either a broad macro level perspective or an in depth micro level perspective. Take for example how our ears detect sound. Our ears first sense using its outer ear, the direction of arrival of sound by some preliminary processing. The ears then face the direction of arrival of sound determined by the macro step, and take in auditory input for further processing to convert the sound into neural signals sent to the brain, which is basically analogous to the micro step. Similarly our eyes also follow two stage micro macro processing to enable vision. It has a peripheral vision which is sensitive to large objects and motion, while the central vision does the micro processing to give detailed information of the scene in focus. Seamless switching between the two stages allows for a very efficient way to solve problems. It leads to not using too much resources throughout, and hence more efficient resource utilization. The second stage which is the micro level processing is triggered only when the first stage gives a go ahead. The first macro stage is always active so as to detect when the micro stage needs to be activated.

The above mechanism yields a resource efficient way to solve many problems in real life. One such area is sensor networks. Collecting data using sensor networks plays quite a big role in many real world scenarios. Concepts such as smart worlds, intelligent products etc. are enabled only due to data that they sense and analyze using which they decide their actions. To show the application of such a mechanism to this field, the source localization problem was taken up, as described in the thesis. The problem of locating the source of sound using three microphones is attempted while using techniques like adaptive sampling, adaptive resolution and inexact design logic. Adaptive sampling basically involves using lower sampling rate during periods of non-activity, analogous to macro stage and switching to a higher sampling rate when a hint of activity is detected. Similarly resolution of the ADC can also be controlled to keep it high when activity or events are detected. Inexact design can also be deployed wherein we reduce the accuracy of the computations as a tradeoff for speed and reduced power consumption by logic circuitry. Usage of all these techniques is analogous to the micro macro stages we described earlier, since we use lesser resources (macro stage) to detect for an actual event starting before devoting more resources (micro stage) like higher sampling rate, higher resolution and computing accuracy.

In this study, a simulation of a scenario for this problem is created using python, and all essential components of the problem like microphones, sound source, analog-to-digital converters and the physics involved are all replicated in the simulation. The techniques were then applied to have an idea of the improvements that they can provide for varying test scenarios like scarce input signal scenarios. The results show considerable reduction in power consumption for a small trade-off on accuracy, and are described in the thesis.

TABLE OF CONTENTS

LIST OF FIGURES

ABBREVIATIONS

SSL	Sound Source Localization
DOA	Direction of Arrival
TDOA	Time Difference Of Arrival
SRP	Steered Response Power
PHAT	Phase Transform
PL	Position Location
PI	Prime Implicant

INTRODUCTION

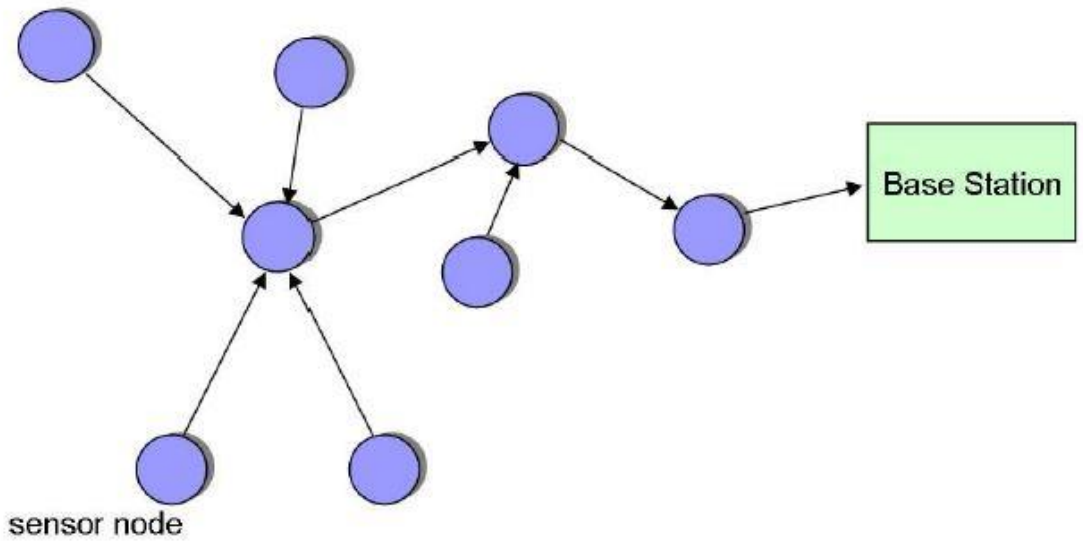
Sensor Networks

Our world is set to be very visibly connected with networks throughout in the near future. Everyday devices will be to sense a broad set of physical phenomena, and this sensing and sharing of information will become a common aspect of small embedded devices. Such devices will talk with one another to organize and coordinate actions. The physical world presents an incredibly rich set of input data like acoustics, image, motion, etc. Using sensor networks it will be possible to observe and sample some of these environmental variables simultaneously at different locations.

A sensor node will have 3 main tasks:

- Sense
- Compute
- Communicate

Sensor Networks find application in monitoring environment and activities. Many a time, sensor networks consist of a large number of nodes. Given the similarity in the data obtained by sensors in a dense cluster, aggregation of the data is an option. That is, a summary or analysis of the local data is prepared by an aggregator node within the cluster, thus reducing the communication bandwidth requirements. (5) .



Aggregation of data increases the level of accuracy and incorporates data redundancy to compensate node failures. A network hierarchy and clustering of sensor nodes allows for network scalability, robustness, efficient resource utilization and lower power consumption.

The Environment

Natural environments are often dynamic and to represent an accurate picture of the changes in the environmental variables, the physical phenomenon is sensed or sampled from the environment at a reasonably fast rate. The physical phenomena measured is basically the determinant of the sampling scale. High-frequency waves typically require higher sampling than phenomena such as temperature, where fluctuations are coarser-grained. The sampling rate is thus a function of both the phenomena and the application. (5) .

Efficiency in sensor networks

Generally, we assume that each node in a wireless sensor network has certain constraints with respect to its energy source, power, memory, storage, and computational capabilities. Not only are the resources of the single sensor nodes limited, but also those of the network as a whole. Especially in wireless sensor networks, which have one shared medium and therefore have to deal with packet collisions, the network capacity is strongly limited. Also, since physical access to the nodes will normally be extremely limited, sensor nodes should work lifelong without human intervention. For such a scenario to happen, we need to be extremely careful regarding power management, which is where the study comes in, as it explores the usage of adaptive sampling, adaptive resolution and inexact design in the activities of wireless sensor networks.

Given a set of network and environment characteristics and definitions, resource consumption (energy and network bandwidth) should be minimized while maximizing the measurement accuracy. The aim is to produce an accurate spatial picture of a certain physical process, while making an efficient use of resources. As events are not uniformly distributed in the environment, not all sensor nodes should collect data samples at a constant static rate.

THREE MICROPHONE PROBLEM

We conceptualized what we named the three microphone problem wherein we use just three microphones to solve various problems in the real world efficiently using our two stage approach. The apparatus at hand is basically the three microphones, a processor or preferably a custom made embedded system for logic processing and control, and Analog-to-Digital converters which can vary their sampling rate and resolution over a given range.

Some of the applications of the three microphone problem that we could identify were:

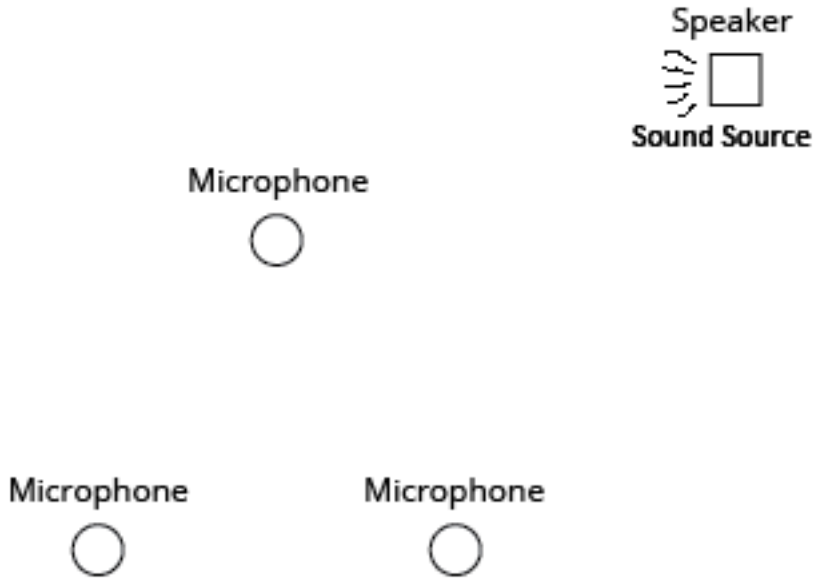
1. **Sound Source Localization**
   Locate a source of sound, such as a machine buzzing and points of turbulence from wind obstruction.

2. **Signal to Noise Enhancement**
   Focus all microphones on a specific point of interest in the room through a delay and sum on all channels to result in constructive interference at the point of interest.
3. **Mixing of Sound**
   Combining multiple sounds into one or more channels.
4. **Noise Cancellation**
   Reducing unwanted sound by the addition of a second sound specifically designed to cancel the first. Multiple microphones make the process more accurate.
5. **Event detection**
   Detect a source of Sound. E.g. speech detection.

The idea is to basically solve one of the problems above using the two stage approach with the help of adaptive sampling, adaptive resolution and inexact design. The solution is extensible to other applications, and can be scaled to scenarios with greater number of sensors also.
I have chosen the source localization problem for the same.

SOURCE LOCALIZATION PROBLEM

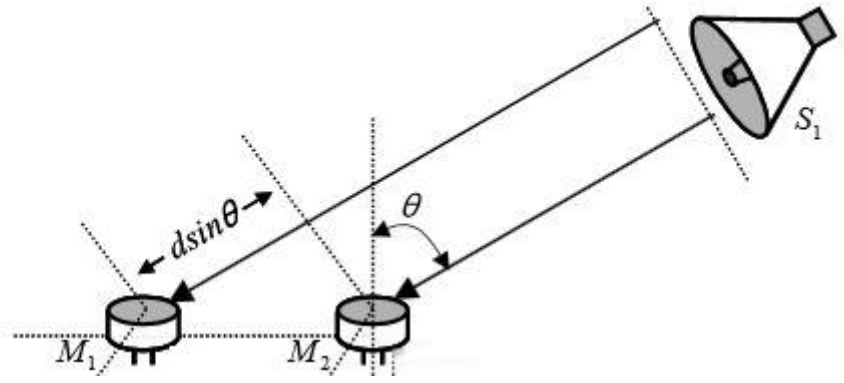
Our aim is to determine location of the sound source by detecting an event while ensuring the whole process efficiently utilizes resources i.e. power. Since we want to do this as part of the 3 microphone problem, we have the constraint of using just three microphones.



The process of determining the location of an acoustic source relative to some reference frame is known as acoustic source localization. Acoustic source present in the near-field can be localized with knowledge of the time difference of arrivals (TDOAs) measured with pairs of microphones.

Direction of arrival (DOA) estimation

Direction of arrival (DOA) estimation of acoustic signals using a set of spatially separated microphones has many practical applications in everyday life. DOA estimates can help decide the direction in which the camera needs to be steered in a teleconferencing room. Direction of arrival (DOA) estimation using microphone arrays uses phase information in the signals picked up by microphones that are spatially separated. When the microphones are spatially separated, the acoustic signals arrive at them with differences in time of arrival. From the known array geometry, the Direction of arrival (DOA) of the signal can be obtained from the measured time-delays. The time-delays are estimated for each pair of microphones in the array. Then the best estimate of the DOA is obtained from time-delays and geometry.



In the figure above, the sound source is assumed to be in the far field of the array. This means that the distance of the source, S, from the array is much greater than the distance between the microphones. Under this assumption, we can approximate the spherical wave front that emanates from the source as a plane wave front. Thus the sound waves reaching each of the microphones can be assumed to be parallel to each other. The signal from the source reaches the microphones at different times. This is because each sound wave has to travel a different distance to reach the different microphones. For example the signal incident on microphone one has to travel an extra distance of dsinΘ compared to the signal incident on microphone two. Using basic geometry, if v is taken to be the speed of sound, the angle Θ is:

$$\Theta = \sin^{-1} \tau v/d$$

Where $\boldsymbol{\tau}$ is the time difference between the arrival of the signals at the two microphones and d is the distance between the two microphones.

Time Delay Of Arrival Estimation (TDOA)

Passive acoustic source localization techniques can be used to locate and track an active talker automatically and hence are able to eliminate the human camera operators in current video-conferencing systems. The problem of passively localizing acoustic sources accurately is difficult. The time delay estimation (TDE)-based localization has become the technique of choice for most people recently. Most practical acoustic source localization schemes are based on time delay of arrival estimation (TDOA) for the following reasons: such systems are conceptually simple. They are reasonably effective in moderately reverberant environments. Moreover, their low computational complexity makes them well-suited to real-time implementation with several sensors.

Time delay of arrival-based source localization is based on a two-step procedure:

- The first stage involves estimation of the time difference of arrival (TDOA) between receivers through the use of time delay estimation techniques. The estimated TDOA's are then transformed into range difference measurements between sensors, resulting in a set of nonlinear hyperbolic range difference equations.

- The second stage utilizes efficient algorithms to produce an unambiguous solution to these nonlinear hyperbolic equations. The solution produced by these algorithms result in the estimated position location of the source.

The TDOA between all pairs of microphones is estimated, typically by finding the peak in a cross-correlation or generalized cross-correlation function. For a given source location, the squared error is calculated between the estimated TDOA's and those determined from the source location. The estimated source location then corresponds to that position which minimizes this squared error. A pair of microphones gives the DOA w.r.t. the axis of the microphones. Since the target has two degrees of freedom, the DOA estimated would give only the direction of the source. On coupling a third microphone with previously installed microphones in an 'L' fashion, the direction of arrival (DOA) w.r.t. the axis containing the third and the center microphone can be obtained. In the second stage, the measured DOA's are used to obtain the source position by solving a system of equations. (4) .

SRP-PHAT Algorithm (Popular Algorithm for TDOA Estimation)

The most straightforward SSL algorithm involves taking a pair of microphones and estimating the difference in time of arrival by finding the peak of the cross-correlation (the direction of arrival is obtained by a geometric transformation from the time of arrival difference and the distance of the mics). For instance, the correlation between the signals received at microphone i and k is:

$$R_{ik}(\tau) = \int x_i(t) x_k(t-\tau) dt$$

The $\boldsymbol{\tau}$ that maximizes the above correlation is the estimated time delay between the two signals. In practice, the above cross-correlation function can be computed more efficiently in the frequency domain.

$$R_{ik}(\tau) = \int X_i(\omega) X_k\ (\omega)\, e^{j\omega t} d\omega$$

To find the $\boldsymbol{\tau}$ that maximizes the value of the correlation, one simple and extendable solution is through hypothesis testing. That is, hypothesize the source at certain location, which can be used to compute. The hypothesis that achieves the highest cross-correlation is the resultant source location. When more than two microphones are considered, we sum over all possible pairs of microphones (including self-pairs) and have:

$$R(s) = \sum_{i=1}^{P} \sum_{k=1}^{P} R_{ik}(\tau_i - \tau_k)$$

This equation is known as the steered response power (SRP) of the microphone array. Again we can solve the maximization problem through hypothesis testing on potential source locations.

To address the reverberation and noise that may affect SSL accuracy, researchers have found that adding a weighting function in front of the correlation can greatly help.

$$R(s) = \psi_{ik}(\omega) \sum_{i=1}^{P} \sum_{k=1}^{P} R_{ik}(\tau_i - \tau_k)$$

A number of weighting functions have been investigated. Among them, the heuristic-based PHAT weighting is defined as

$$\psi_{ik}(\omega) = 1/|X_i(\omega)||X_k(\omega)|$$

PHAT has been found to perform very well under realistic acoustic and reduces the complexity from $P^2\ to\ P.T$his algorithm is called SRP-PHAT. (2, 8).

Hyperbolic Position Location Of Source

The time-delay of the sound arrival gives us the path difference that defines a hyperbola on one branch of which the emitter must be located. At this point, we have infinity of solutions since we have single information for a problem that has two degrees of freedom. We need to have a third microphone, when coupled with one of the previously installed microphones, it gives a second hyperbola. The intersection of one branch of each hyperbola gives one or two solutions with at most of four solutions being possible. Since we know the sign of the angle of arrivals, we can remove the ambiguity.

Hyperbolic position location (PL) estimation is accomplished in two stages. The first stage involves estimation of the time difference of arrival (TDOA) between the sensors (microphones) through the use of time-delay estimation techniques. The estimated TDOAs are then utilized to make range difference measurements. This would result in a set of nonlinear hyperbolic range difference equations. The second stage is to implement an efficient algorithm to produce an unambiguous solution to these nonlinear hyperbolic equations. The solution produced by these algorithms result in the estimated position location of the source. The following sections discuss the techniques used to perform hyperbolic position location of the sound source. Accurate position location (PL) estimation of a source requires an efficient hyperbolic position location estimation algorithm. Once the TDOA information has been measured, the hyperbolic position location algorithm will be responsible for producing an accurate and unambiguous solution to the position location problem. Algorithms with different complexity and restrictions have been proposed for position location estimation based on TDOA estimates.

When the microphones are arranged in non-collinear fashion, the position location of a sound source is determined from the intersection of hyperbolic curves produced from the TDOA estimates. The set of equations that describe these hyperbolic curves are non-linear and are not easily solvable. If the number of nonlinear hyperbolic equations equals the number of unknown coordinates of the source, then the system is consistent and a unique solution can be determined from iterative techniques. For an inconsistent system, the problem of solving for the position location of the sound source becomes more difficult due to non-existence of a unique solution. (4) .

Adaptive Sampling

Sampling

Most physical phenomena are widespread in time and space. Thus, they can be analyzed in one or both these domains. For example a certain point in space can be considered and the time dependency of the signal at this point is analyzed. However, there also exists a space domain, which can only be examined with multiple sensors at different positions in space.

Time domain

In order to properly sample an analog signal the Nyquist-Shannon sampling theorem must be satisfied. In short, the sampling frequency must be greater than twice the bandwidth of the signal (provided it is filtered appropriately):
$fs \leq 2Bn$
fs = sampling frequency
Bn = network bandwidth
With adaptive sampling the sampling interval may vary over time and adapts to the signal complexity. But how can the sensor nodes determine the correct sampling rate:

We could do one of three:

- A central server takes part in this decision process
- Each sensor node determines on its own the correct sampling rate.
- Sensor nodes communicate with server only from time to time and adapt sampling rate most of the time autonomously.

When measuring a signal, a minimal accuracy should always be guaranteed. This can easily be achieved by using the naive solution of oversampling. An oversampled signal can be defined as a signal, which is sampled at a higher rate than necessary for the representation of the signal bandwidth.

Sensor nodes using oversampling however will run into several problems:

- Most of the sampled data must be transmitted to a central server. The CPU at the central server might have to process unnecessary data from numerous sources, but this would not affect the result significantly.
- Unnecessary data is transmitted through the communication channel. Moreover, in cases of low bandwidth networks the option of oversampling might not be available at all.
- Power conservation is critical for sensor nodes. Oversampling leads to increased power consumption because of increased data transmission and due to more power used for sampling itself. (1) .

Thus, oversampling is not a suitable solution for sensor networks. The goal of an adaptive sampling approach is to make the rate of sensing dynamic and adaptable:
As signal complexity grows, the sampling interval is scaled down and vice-versa.

Another way of using adaptive sampling is by sampling in the space domain. Many a time, there arises opportunity to use the spatial arrangement of the sensors to our advantage, and thus save on continuous operation of all sensors by extrapolating data obtained at one sensor during low activity periods and using all sensors at high sampling rate during high activity periods. We have explored this technique too in our simulations.

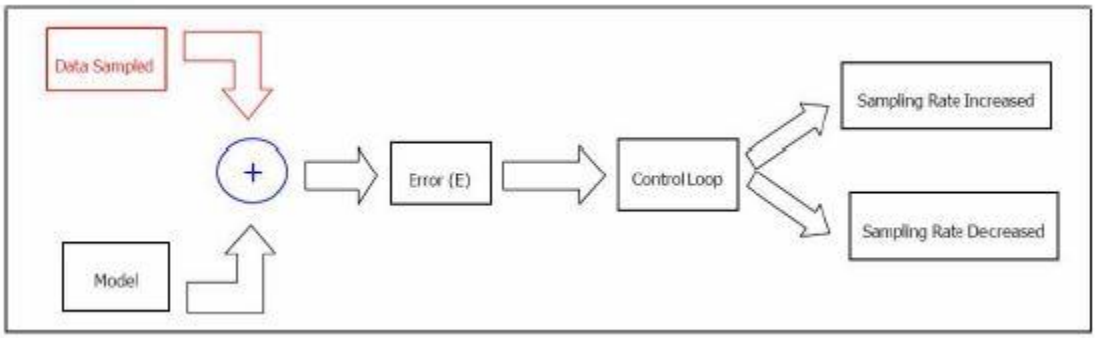
Adaptive Sampling applied to the source localization problem

Completely autonomous nodes: Feedback Control Mechanism

Each individual node adopts a feedback control mechanism in order to make the rate of sensing dynamic and adaptable. Figure presents the feedback control mechanism proposed.

With an internal model representing the environment, each node can make predictions of future measurements. The real sampled data will get compared to these model predictions and an error value will be calculated on the basis of the comparison. If the error value is more than the

predefined margin of error, then the node will collect the data at higher sampling rate, and if it is lower, the sampling rate will be decreased. Each node decides on its own how to adapt the sampling rate. The adaptation works completely autonomous.



Control mechanism used internally within the nodes to control the sampling rate.

Basic notion is to vary the sampling rate to adapt to certain conditions defined by us in order to reduce power consumption due to sampling. Most scenarios need high resolution data only during abnormal or unique circumstances. The data during the rest of the time can be approximated by previous data to a satisfactory extent.

Let us try to use adaptive sampling on a general scenario. In most cases, the above notion is true. What we need to understand is how to realize is when to switch to high sampling rate from the normal sampling rate. We can use a metric to decide this.

Metrics:

1. If the variance of the input data is low and the mean of the incoming input data is close to historical mean, we need not switch to higher sampling rate.

2. Another metric can be: If currently observed data point is close to historical maximum or minimum, it has a high chance of being an important data point. So we need to switch to high sampling rate.

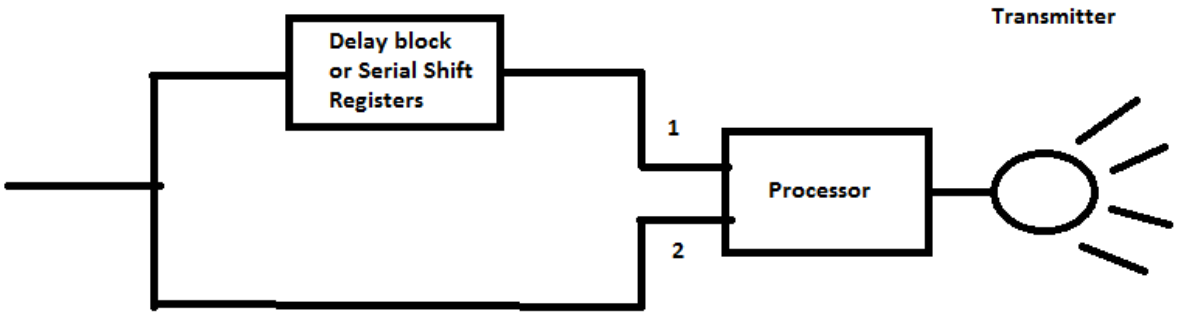
The normal or average rate of sampling is decided by the min rate needed to decide if event or data is important. Also, time needed to recognize the importance of the event. The sampling rate should also be of a reasonable value that will be enough for the metrics discussed above to make sense. Basically the requirement decides the way to choose the average sampling speed.

Loss of data during the period of switching from low to high sampling rate:

Another consideration that needs to be addressed is whether we lose data in the time we need to recognize that sampling rate needs to be switch to a higher value. Let's call this time T.

There are 2 ways to address this:

1. Data points for the last T time should be stored in a buffer. A buffer is a serial shift registers to hold last n data points which must be designed according to purpose.
The output should switch to this buffer once it recognizes the need for higher sampling rate. The buffer should store data for last time T data points.

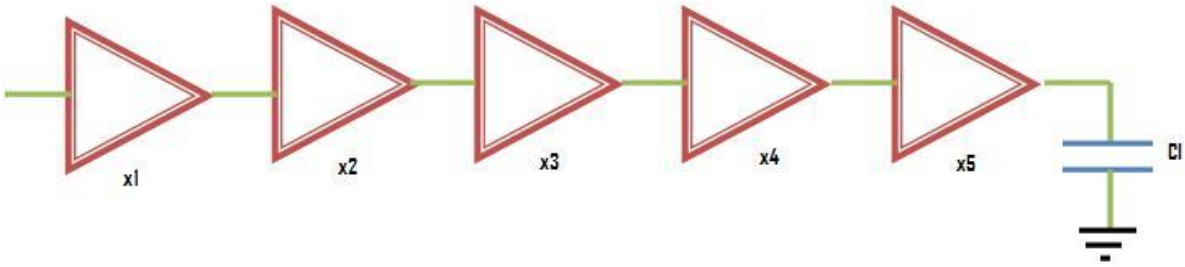


2. Add a delay module that delays for time T. On the original line, if using our metric we detect a need to switch to high sampling, we switch to the delayed line. The delay module to use will depend on two things:

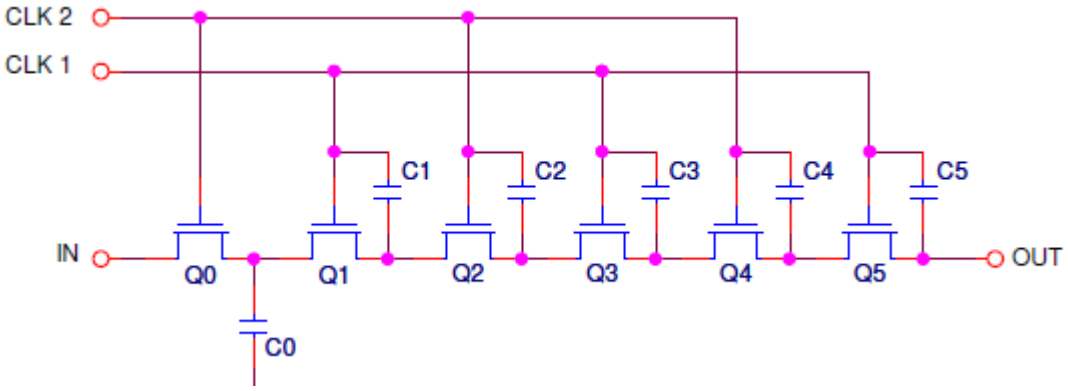
- How much delay we are talking about
- What level of distortion in data points can we be fine with?

Delay module:

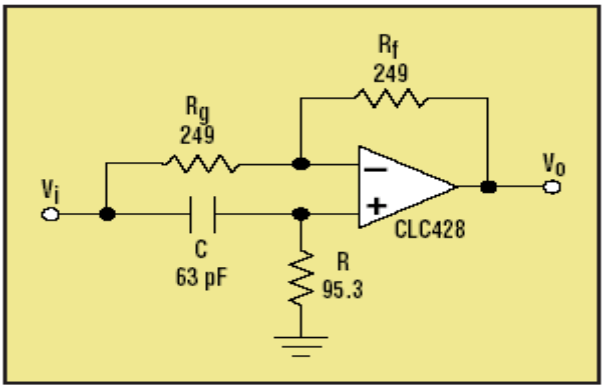
If delay needed is in range of nanoseconds, appropriate number of cascaded inverters can be used to get required delay.



If we needn't require high frequency components, we can use series of capacitors sections to get the required delay. This way of achieving a delay is also called bucket brigade.



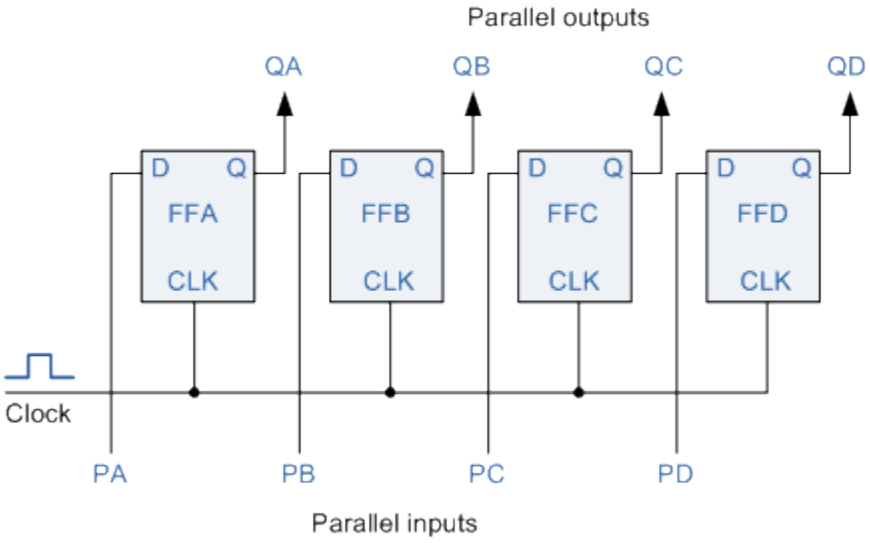
We can also use Pode's formula and get an approximation of the delay using active components. This again will cause some distortion.



This compact circuit provides a controllable analog delay that can be accurately predicted if an op amp with a wide bandwidth is selected.

If we cannot compromise on quality of data points, we should use the serial shift register queue for the purpose of storing data.

Here delay will be digital and there will be no loss of quality. However, we need to sample and get data in digital domain before applying the delay in this way.



Without delay module

Another way to try and get delayed data is by actually placing the multiple sensors apart at calculated distances. E.g. For the source location problem of the 3 microphones, if the microphones are placed such that time taken for sound to travel to each microphone differs by time T delay that we need, we can create a switching circuit that can make user of this data to switch to the latter microphone and start sampling at high rates with high resolution.

Adaptive sampling in the space domain.

We can also perform adaptive sampling in the space domain. We don't always sample inputs from all microphones, instead, we keep a low threshold on a few dispersed microphones, which when reached will signal other microphones to come into normal sampling mode which will alert all other microphones to start at low sampling rate. When you reach a higher threshold, all microphones go to high sampling mode.

Adaptive Resolution applied to source localization problem

Again, we will look into using resolution of ADC that adapts to the current situation. The same logic to decide sampling rate applies to resolution also. We use a high resolution Adc during high sampling rate which is when data is termed to be important. We switch to low resolution ADC the rest of the time.

Inexact Design

Over the last decade, there has been an increasing interest in innovations in realizing "good-enough" and parsimonious systems that could afford an *imperceptible* degradation in their output quality if they could get significant resource-savings (either energy consumption, critical path delay and/or silicon area) in exchange. This was driven in part by the necessity to innovate and sustain the technology scaling (and as a result, the resource efficiency) prophesied by the Moore's law which was facing the hurdles of process- and parameter-variation induced errors as well as the forecasts that a large part of the emerging applications would involve Recognition, Mining and Synthesis (RMS) workloads, most of which exhibit varying levels of error resilience. In these systems, any error (either caused probabilistically due to inherent device variations or perturbations, or induced deterministically) can be viewed as a commodity that can be exchanged for substantial savings in hardware cost (energy, delay and/or area) *without* the necessity of any error correction mechanisms.

Inexact circuits are parsimonious or "adequately engineered" in terms of (physical) implementation and cost (quantified through energy, delay and/or area metrics) much lesser than their "over-engineered" conventional correct counterparts while achieving the needed quality of

output for a target application. In this section, we aim to assess the previously proposed physical-level (e.g. voltage over scaling) and architectural-level (e.g., probabilistic pruning and probabilistic logic minimization[6] techniques by coining a term called potential lowering, broadly referring to a lowering of the energy configuration.

Defining Error Metrics

We can broadly classify error-resilient applications into two types: ones which have a bound on the total number of erroneous computations (such as the number of incorrect memory address computations in a microprocessor) and others (such as the computation of the value of a pixel by a graphics processor) which have bounds on the magnitude of error.

Error Rate = Number of Erroneous Computations/Total Number of Computations

Physical level inexactness:

Most of the initial work in realizing inexact hardware systems focused on leveraging the supply voltage as the control knob for gleaning the quadratic energy savings made possible through *voltage overscaling*.

$$P \propto V^2$$

So reducing V at the expense of accuracy leads to significant improvements in power consumption. In most cases, especially where transistors are involved, reducing V still allows for almost 100% accuracy, since the mode of operation of the transistors still guarantees accurate input.

Architecture level inexactness:

Probabilistic Pruning is an architecture- or logic-layer technique through which we "prune" or delete computational blocks and their connecting wires from a fully functional circuit design. [1] The criterion for deciding that a block can be removed is based on the significance that the node has in contributing to the output value, and also its activity level when the circuit is exercised using a canonical set of inputs. For example, a node whose output has a lot of significance but is pretty dormant across most inputs of interest (determined by the application workload's distribution in general, assumed to be uniform in our examples) is a candidate. Similarly, a very active node can be a candidate for pruning if the values it computes do not have significant contribution to the outputs.

The probabilistic pruning algorithm consists of two main functions:

- *Ranking Function.* The goal of the ranking function is to rank the nodes based on their *value* determined by the *Significance-Activity Product* (SAP) metric.
- *Pruning Function.* The goal of the pruning function is to iteratively prune the nodes with the least SAP values until the target error bound is reached. (1, 3) .

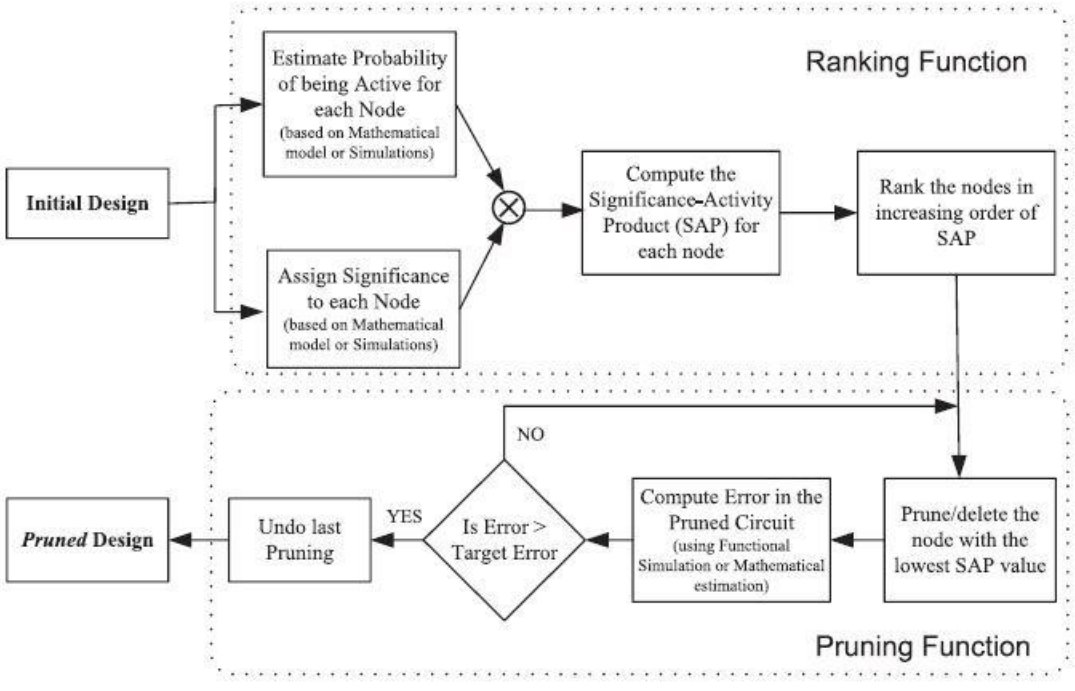
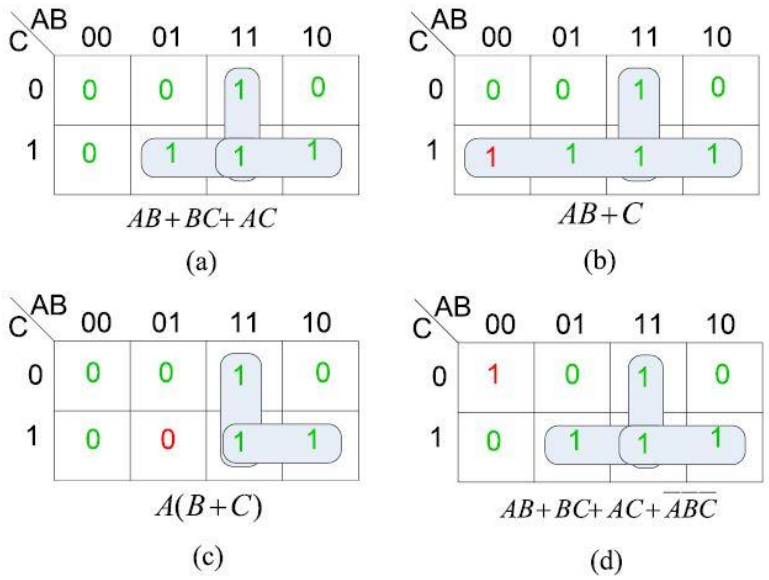


Diagram referenced from (1) .

Logic level inexactness:

Probabilistic logic minimization is a logic-level technique wherein we systematically minimize circuit components (or nodes) guided by the significance and the input combination probabilities of those nodes while staying within the error boundaries dictated by the application. Similar to probabilistic pruning, it is carried out at the design level and hence, incurs zero hardware overhead. The proposed algorithm takes advantage of the notion of introducing *bit flips* in the minterms of boolean functions in the context of fast error detection and subsequently used for enhancing circuit yield and for designing low-power inexact circuits. The key to the probabilistic logic minimization algorithm is the notion of introducing bit flips in the minterms of boolean functions to further minimize, thereby achieving gains (energy/area/delay) through literal reduction while causing an error due of such bit flip(s). However, not all bit flips of minterms would result in expanding the prime implicant (PI) cubes, and some of them might result in negative gains. Hence, it is important to identify the "favorable" bit flips (or the bit flips which further minimize the function) and discard the unfavorable ones. To illustrate through an example, Figure (a) below shows a function (Carry logic) that is widely prevalent in most datapath elements. Assuming that the application would only be able to tolerate at most one bit flip at this logic function (probability of error = 1/8), Figures (b) and (c) give examples of favorable 0 to 1 and 1 to 0 bit flips, respectively, as they minimize the logic function, whereas Figure (d) shows an unfavorable bit flip leading to an increased logic function complexity. Another alternative for addressing unfavorable minterms is by using *don't care* (DC) conditions in the Karnaugh maps. Hence, we can conclude that the introduction of favorable bit flips would lead to further minimization of a logic function owing to the expansion of PI cubes, thereby

achieving cost (energy, area, and delay) gains at the expense of error, which is proportional to the number of such bit flips introduced. (10) .



Example of k-maps of a) initial correct function b) function with favorable 0 to 1 bit flip c) function with favorable 1 to 0 bit flip d) function with unfavorable 0 to 1 bit flip

Referenced from (10) .

Inexact design applied to source localization problem

In the source localization problem, to determine the TDOA estimates, we use the SRP-PHAT algorithm. The SRP calculation step of the algorithm is basically

$$R(s) = \sum_{i=1}^{P} \sum_{k=1}^{P} R_{ik}(\tau_i - \tau_k)$$

We then add a weighting function $\psi_{ik}(\omega) = 1/|X_i(\omega)||X_k(\omega)|$ to this equation to reduce the complexity of the algorithm. So we already have some inexactness because of the heuristic weighting function we added.

We can go a step further and use an inexact adder for the addition of the pairwise cross correlation equations between microphones. This inexact adder is obtained by using the probabilistic pruning technique described above on an adder circuit.
For e.g.

After applying the ranking function described in the above technique on a 16 bit adder and then pruning the least significant nodes based on these ranks, we can reduce the precision to 14 bits. Healing is then done which takes the pruned circuit and reconnects the blocks that might have become isolated due to the pruning in the previous step. The isolated nodes are either set to 0 or 1 based on which of the two options leads to a lower relative error in most scenarios.

From reference paper 3 in the references section, we can obtain values of accuracy and power improvement from such a pruning.

Reducing the precision of the adder to 14 bits causes a relative error of 0.007% but has a gain of 1.334X in power-density-area. (3) .

Using these adders for the SRP-PHAT algorithm, which already uses a heuristic helps reduce power consumption at very little loss of accuracy, and this is especially helpful in a scenario when we use an embedded system with inexact adders as the logic component of a wireless sensor network.

SIMULATIONS

Assumptions:

We assume the following conditions under which location of sound source is estimated:

- Single sound source, infinitesimally small, omni directional source.
- Reflections from the bottom of the plane and from the surrounding objects are negligible.
- No disturbing noise sources contributing to the sound field.
- The noise source to be located, is assumed to be stationary during the data acquisition period.
- Microphones are assumed to be both phase and amplitude matched and without self-noise.
- The change in sound velocity due to change in pressure and temperature are neglected. The velocity of sound in air is taken as 330 m/sec.
- Knowledge of positions of acoustic receivers and perfect alignment of the receivers as prescribed by processing techniques.

Description of Simulation:

Microphones, variable analog-to-digital converters, sound emitting sources (transmitters), a processor with computing capabilities and physics behind sound travel and power consumption is modelled using python.

Sound sources are made active at random locations in the modelled xy space, emitting sound of random amplitudes for randomly generated timings. An event is defined in our simulation as a random long signal emitted by a sound source with a reasonably high amplitude.

What we attempt to do in our simulation is to use minimum power but be able to capture maximum data i.e. try to not miss any events that happen in such a scenario of sound monitoring. In this regard, we attempt to use adaptive sampling, adaptive resolution and inexact design to improve our power consumption without losing much accuracy in data collection.

Moving average filter:

Given a series of numbers and a fixed subset size, the first element of the moving average is obtained by taking the average of the initial fixed subset of the number series. Then the subset is modified by "shifting forward"; that is, excluding the first number of the series and including the next number following the original subset in the series. This creates a new subset of numbers,

which is averaged. This process is repeated over the entire data series. This is basically applying the moving average filter on data already present.

We use the moving average of the last few samples obtained at each microphone in real time to decide if the mean of the incoming input data is close to historical mean or not. If it isn't close to the historical mean, we switch to a higher sampling rate and resolution, else we continue with the same earlier sampling rate. Switch down to a lower rate also is similar, as once activity stops, the moving average filter gives a lower output value which in turn signals the processor to reduce the sampling rate of the ADC in the microphone.

Feedback control mechanism:

What we are doing using the moving average filter is essentially feedback control. We have a model, in our case of some minimal average noise in the background. Once the received input data deviates from this model, like an event occurs which changes the output of the filter, we increase sampling rate since the error calculated with respect to the model increases indicating activity.

Sampling in space domain:

In the simulations used, sampling in space domain was also used. The input received by one of the microphones is used as a metric to decide if the other microphones need to be sampling right now or can be in sleep. A smaller threshold than the high sampling rate threshold is decided, and once the threshold is breached i.e. there is a deviation from the expected model of input, the ADC's of the other microphones, which were in sleep until now, are made to sample at their normal sampling frequency. Once the higher threshold or model error needed to activate the highest sampling rate is reached, all the microphones are set to this state in order to record data of the event in question.

RESULTS

Simulations for various scenarios with events and random noise generation in background were run. The results show a significant reduction in power consumption by the use of adaptive sampling and resolution. The general frequency of events and background noise is what determines how much improvement our techniques can provide. The improvements will be maximum in scenario's where events occur only once in large time periods.

3 of the simulations that were attempted are detailed below. The first two simulations use adaptive sampling and resolution in the frequency domain. The third simulation adds the dimension on adaptive sampling in space domain also to further improve results.

Simulation Scenario	Details
1	Locating the sound source with **dual event occurrences** and **minimal background noise** during the time frame while using **adaptive sampling in time domain** and **adaptive resolution**
2	Locating the sound source with a single event occurrence and excessive background noise during the time frame while using **adaptive sampling in time domain** and **adaptive resolution**
3	Locating the sound source with **dual event occurrences** with **excessive background noise** during the time frame while using **adaptive sampling in time and frequency domain** and also **adaptive resolution**

Simulation Output for each scenario:

Simulation Scenario	% of time at 1x sampling rate/resolution*	% of time at 10x sampling rate/resolution *	% of time at 100x sampling rate/resolution *	Power reduction assuming (P α sampling rate) *	Loss of event data due to time needed to switch to 100x resolution *
1	0	82%	18%	73%	5%
2	0	80%	20%	72%	2%
3	50%	10%	40%	60%	5%

* All values are taken as an average across all three ADC's involved.

If simulation 3 did not take into account adaptive sampling in space domain, Power reduction would have been 54% instead of 60%.

To reduce loss of event data, any of the methods like adding a delay block or using a circular buffer to store incoming data as described under the section "Loss of data during period of switching from low to high sampling rate" in adaptive sampling applied to SSL problem can be used.

If using the pruning technique on adders that solve the SSL problem as described in section "Inexact design applied to SSL", Power reduction achieved is an additional 30% with an accuracy loss of 0.02% due to the inexactness of the circuitry.

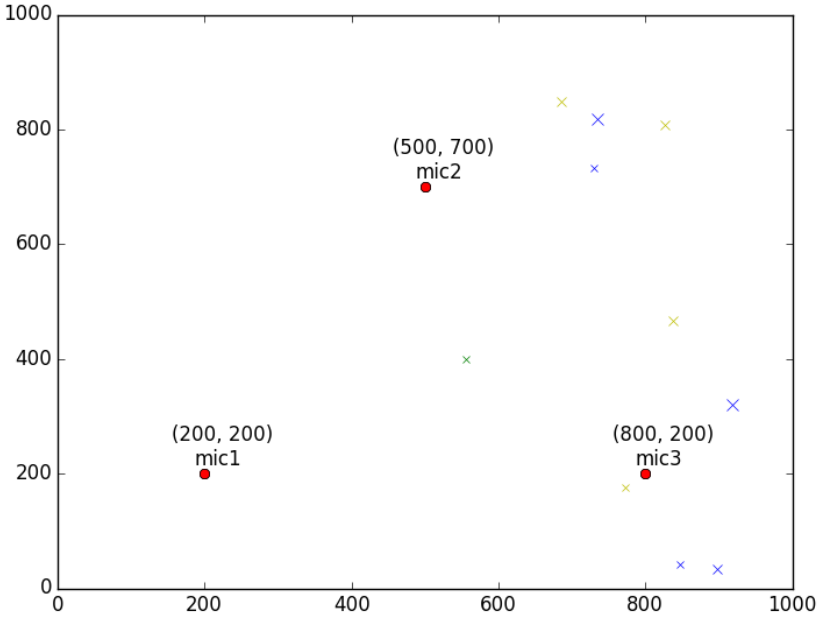
Simulation Scenario 1:

Two events, each with reasonable magnitude and time duration happen during the simulation period.

There is reasonable amount of noise generated in the background.

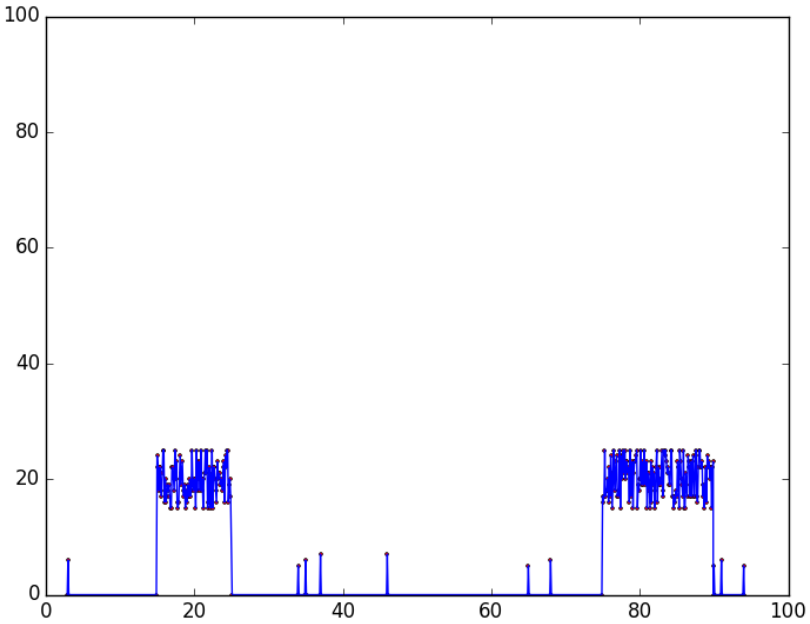
Simulation 1 graphs:

Microphones locations and location of sound source that emitted the random sounds generated for the simulation represented by x marks

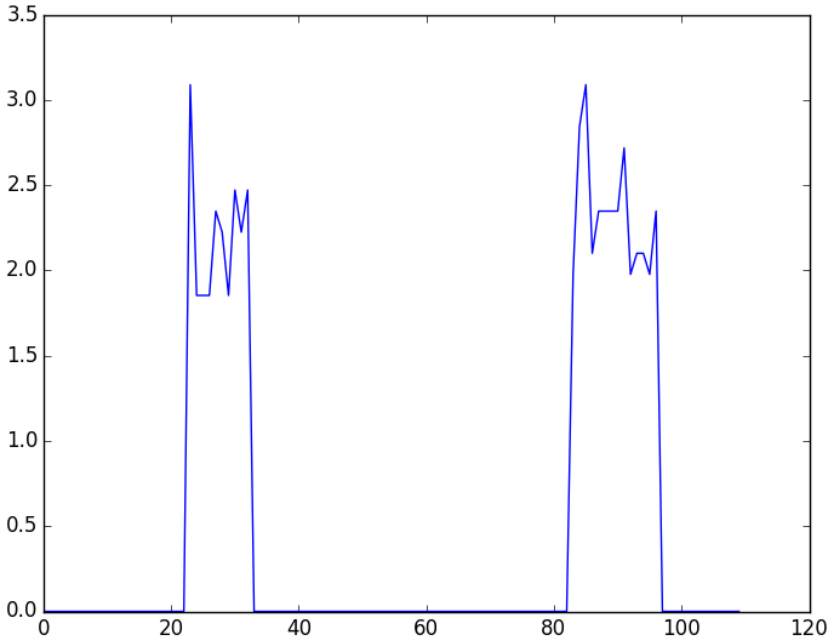


.

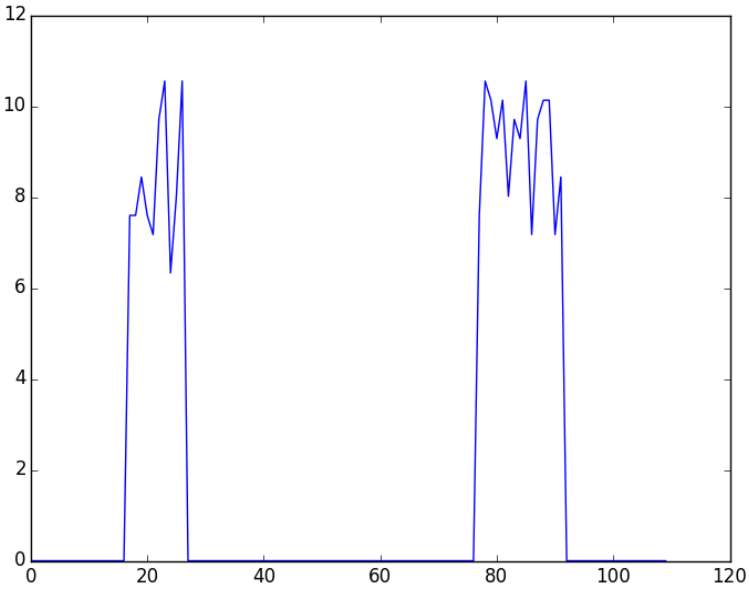
Random sound signals as generated as input for the simulation



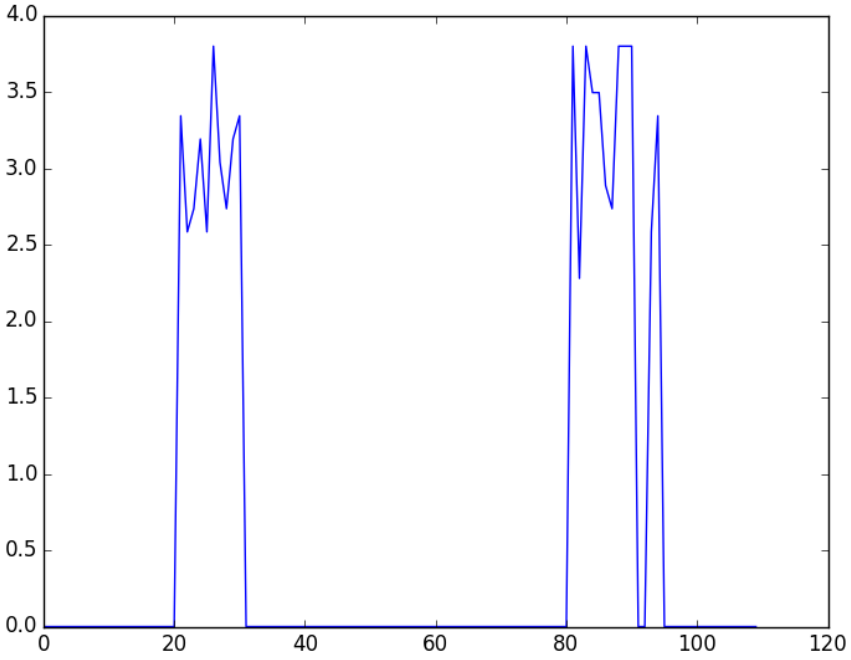
Input sound as received by microphone 1



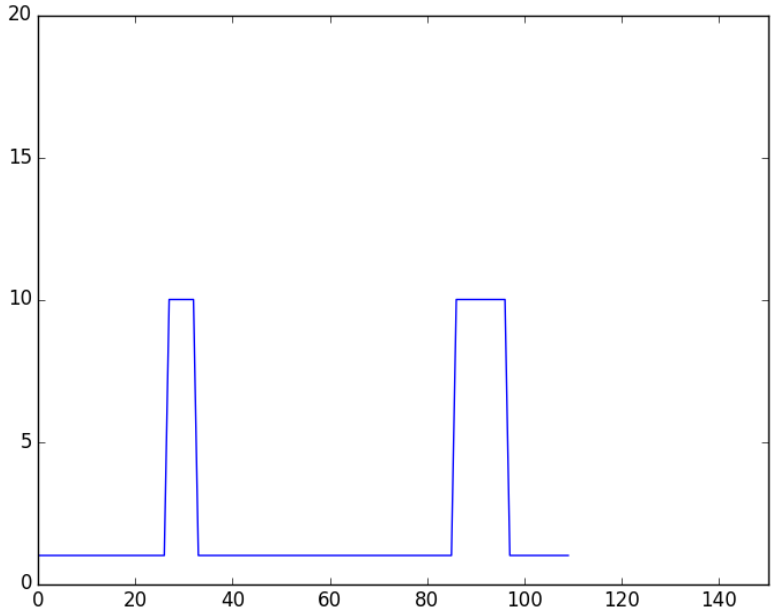
Input sound as received by microphone 2



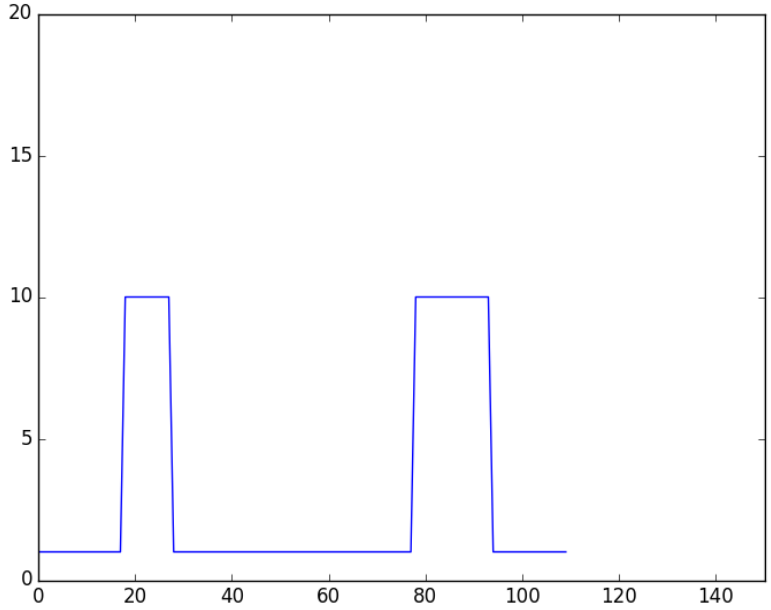
Input sound as received by microphone 3



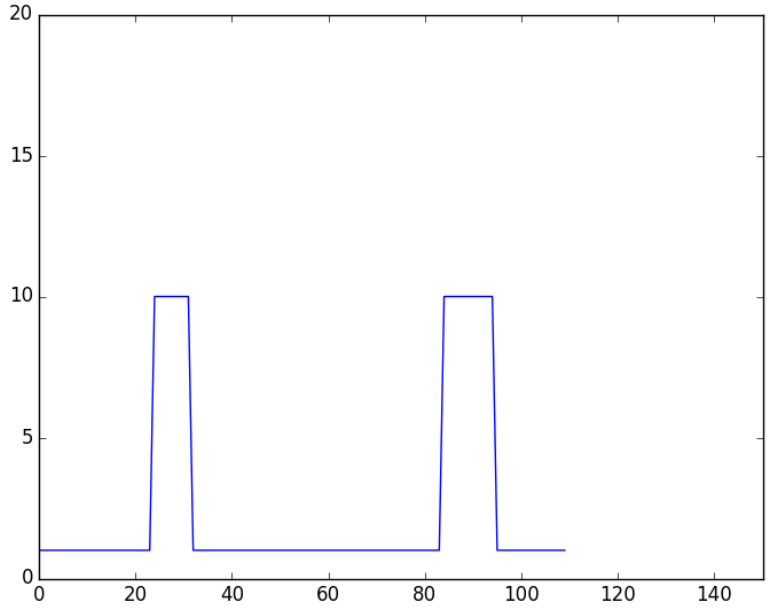
Adaptive Sampling Rate of ADC1



Adaptive Sampling Rate of ADC2



Adaptive Sampling Rate of ADC3

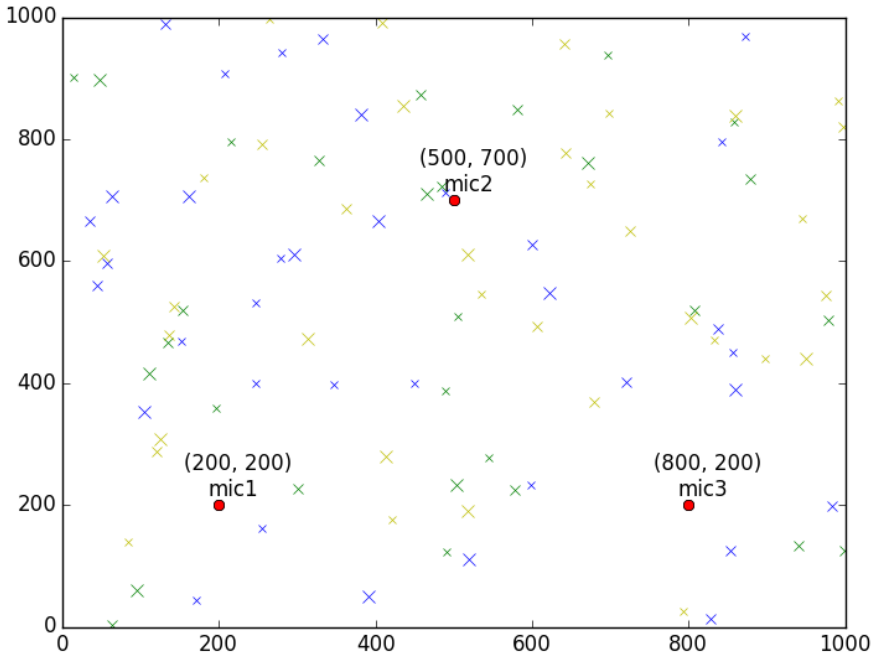


Simulation Scenario 2:

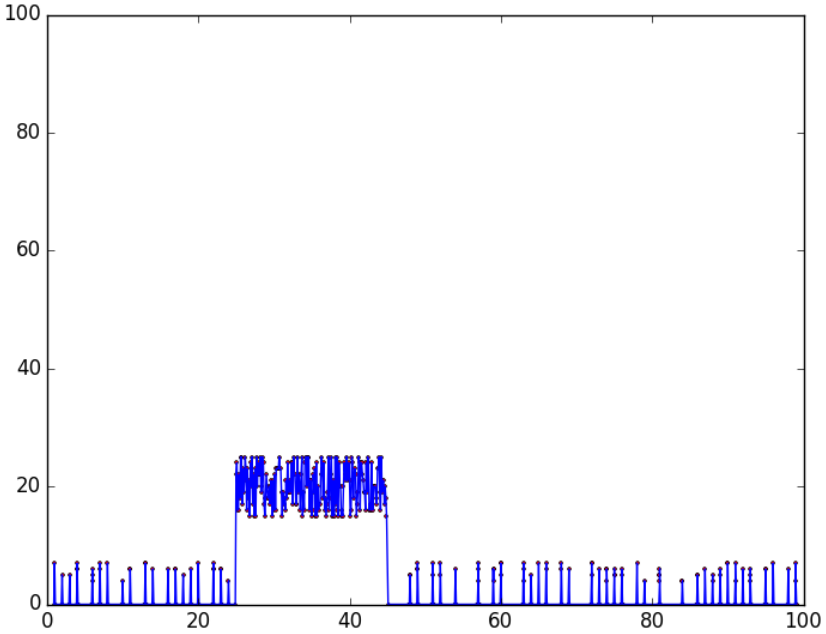
- Single event with reasonable magnitude and time duration
- Excessive background noise generated.

Simulation 2 graphs:

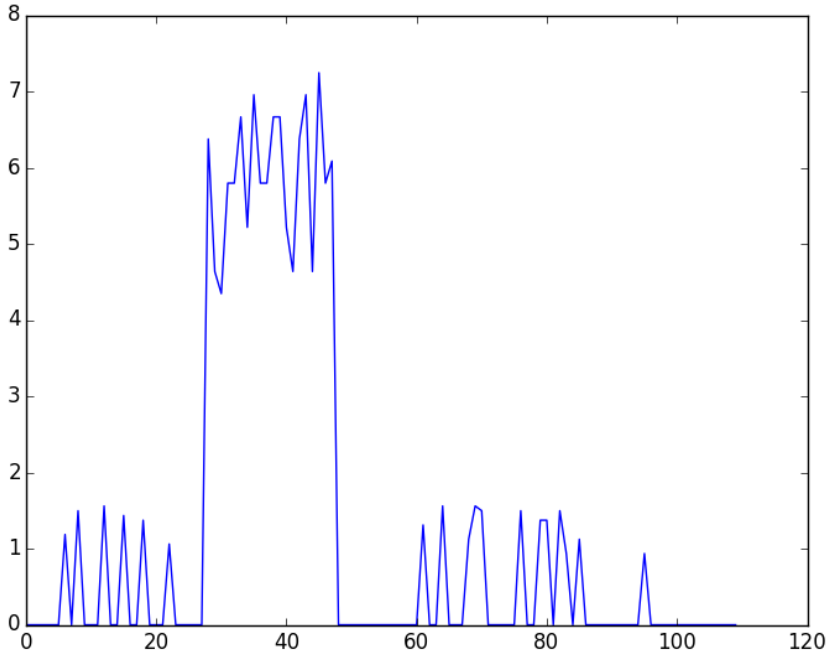
Microphones locations and location of sound source that emitted the random sounds generated for the simulation represented by x marks



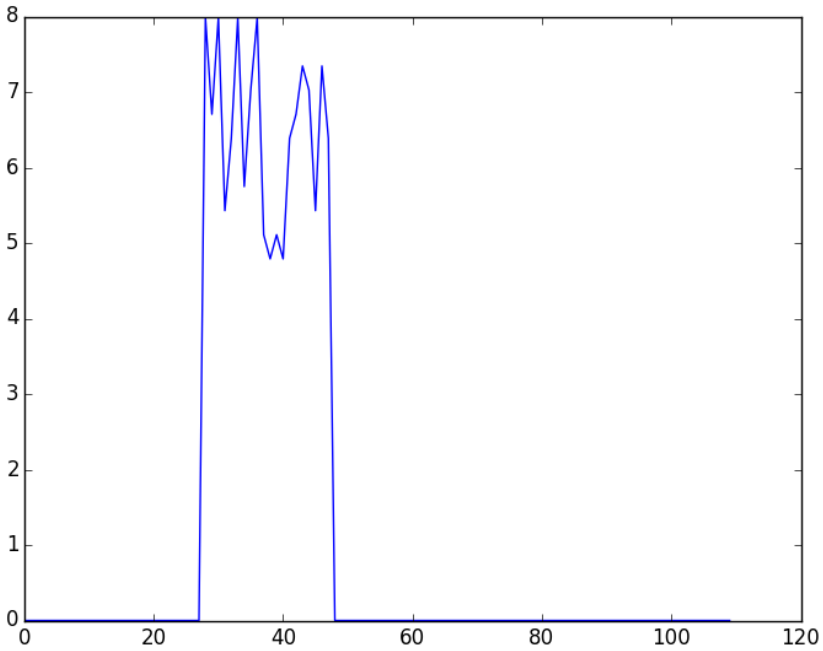
Random sound signals as generated as input for the simulation



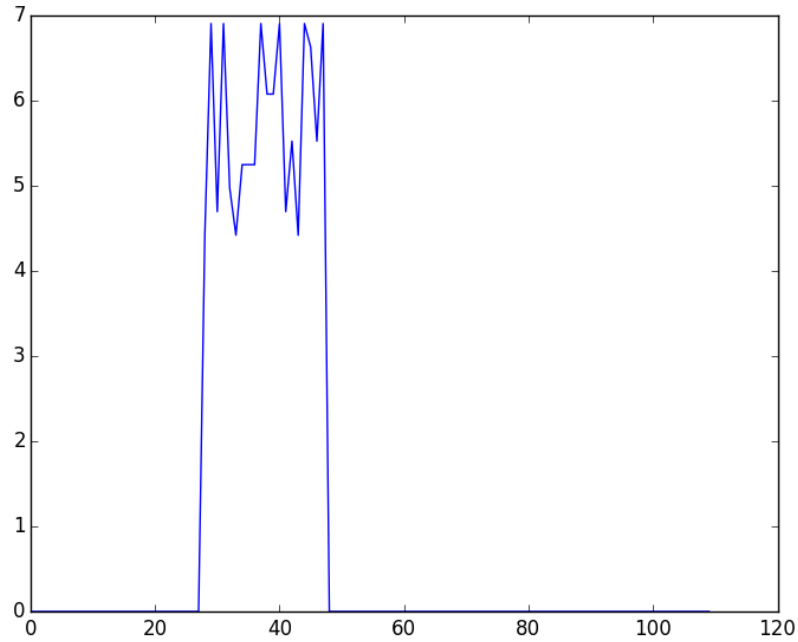
Input sound as received by microphone 1



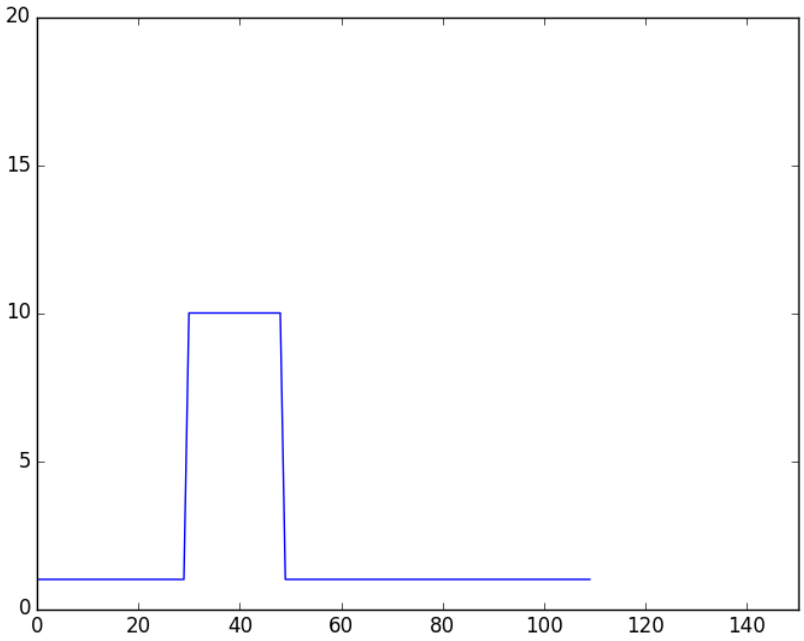
Input sound as received by microphone 2



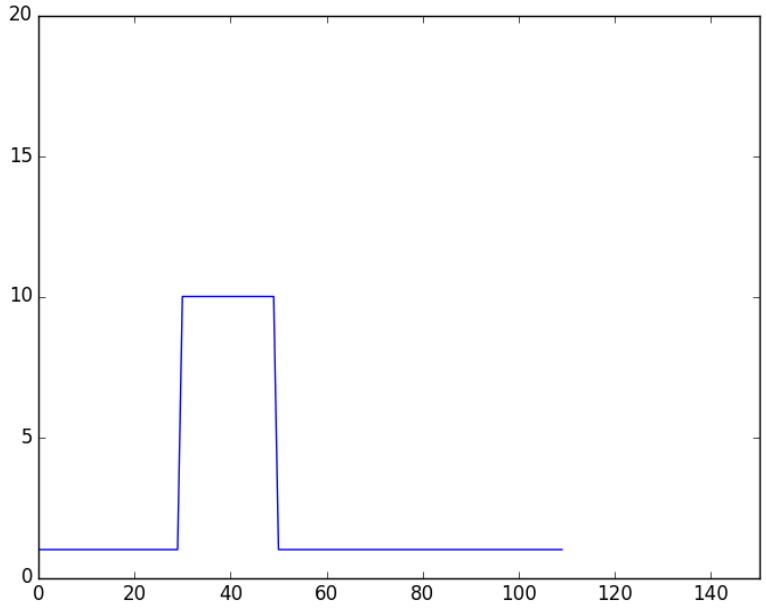
Input sound as received by microphone 3



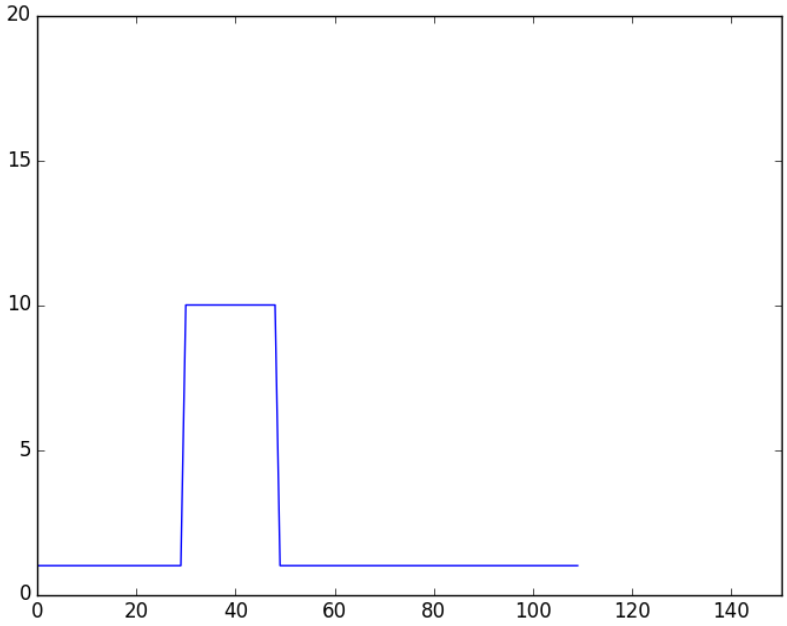
Adaptive Sampling Rate of ADC1



Adaptive Sampling Rate of ADC2



Adaptive Sampling Rate of ADC3



Simulation Scenario 3: (includes results using adaptive sampling in space domain also)

Two events, each with reasonable magnitude and time duration happen during the simulation period.

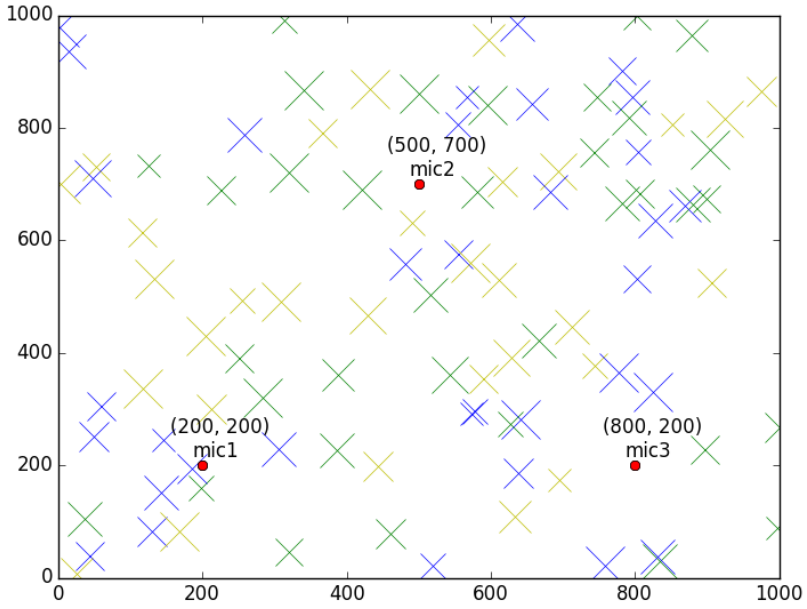
Excessive noise is generated in the background.

ADC's connected to outputs of microphones are in sleep mode when negligible activity is present. Only ADC of one microphone is kept at normal sampling rate, which when it senses some activity of a minimal threshold, signals the processor to get all the other sleeping ADC's to normal sampling rate. If activity increases further, the sampling rate of the ADC's is further increased to a higher value to get more fine grained input data related to the event in question. Resolution of the ADC is also changed in a similar way as sampling rate.

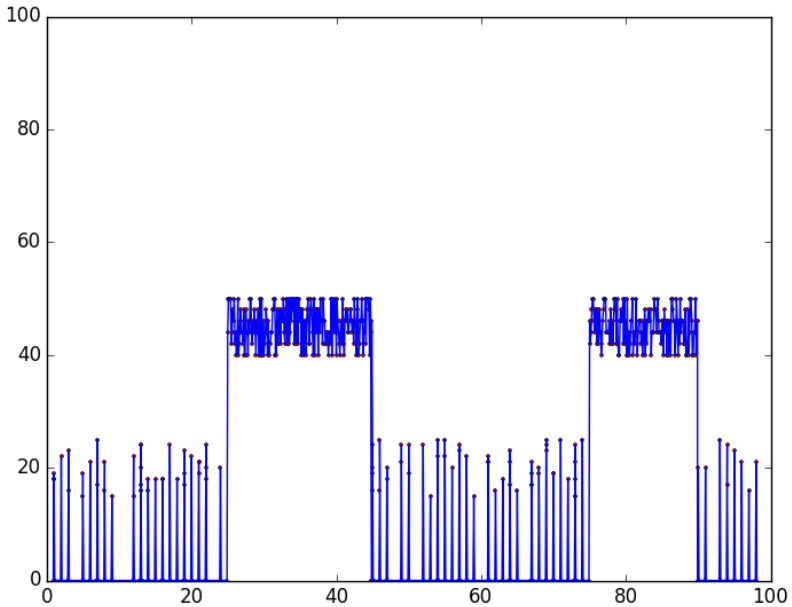
The graphs show a sampling rate of 1 for sleep mode, 10 for normal sampling rate and 100 for high sampling rate condition in the ADC sampling rate graph.

Simulation 3 graphs:

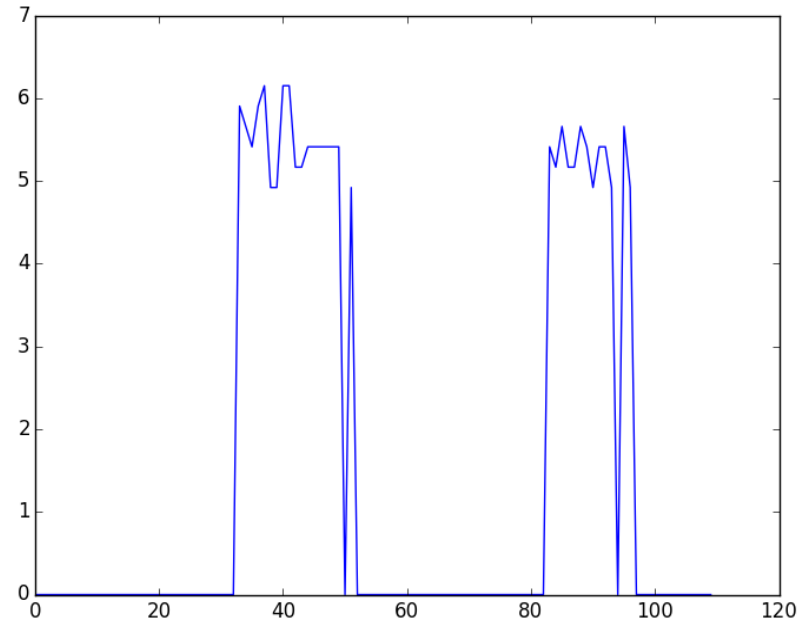
Microphones locations and location of sound source that emitted the random sounds generated for the simulation represented by x marks



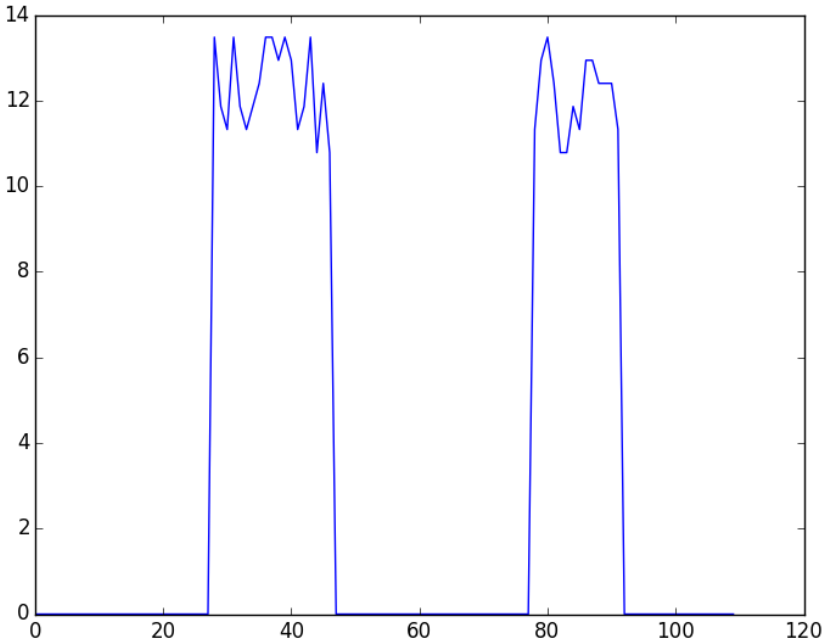
Random sound signals as generated as input for the simulation



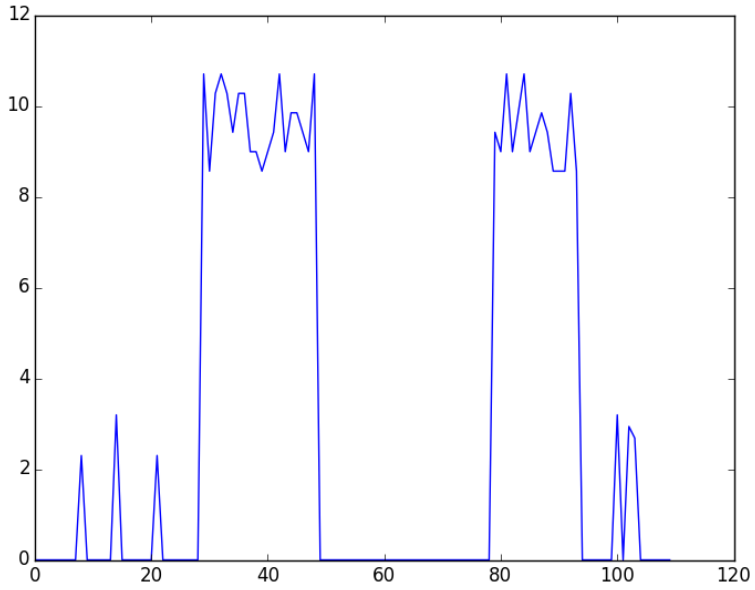
Input sound as received by microphone 1



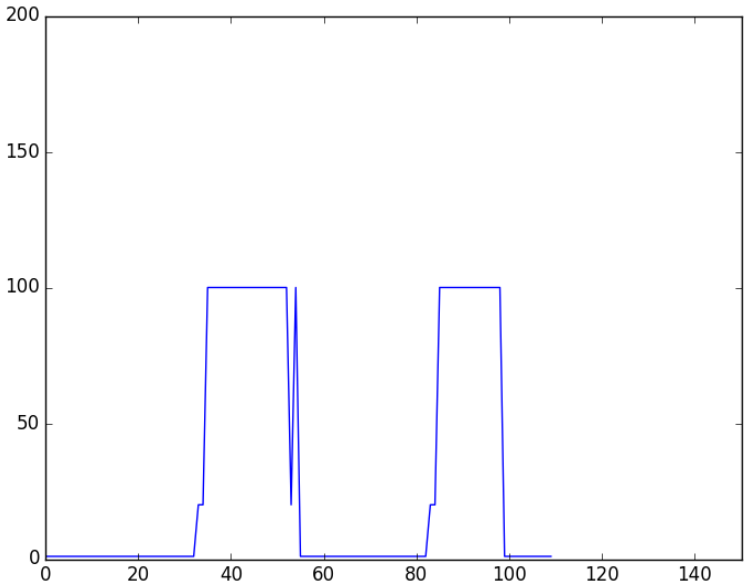
Input sound as received by microphone 2



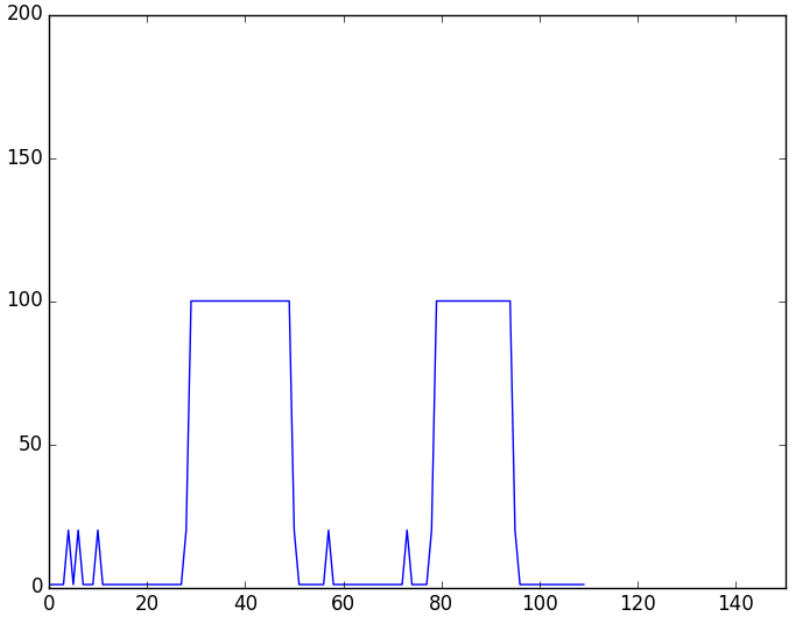
Input sound as received by microphone 3



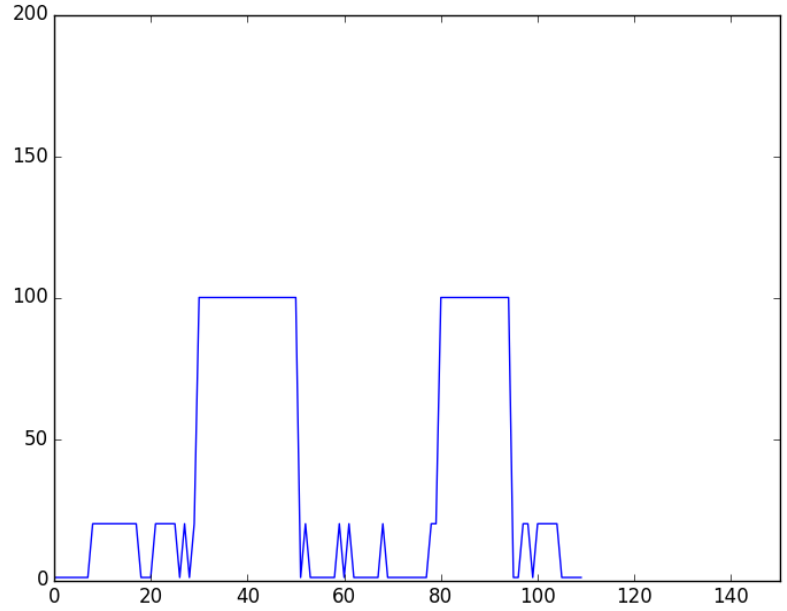
Adaptive Sampling Rate of ADC1



Adaptive Sampling Rate of ADC2



Adaptive Sampling Rate of ADC3



CONCLUSION

Adaptive sampling, adaptive resolution and inexact design are techniques that when used provide great reductions in power consumptions, especially in sparse input signal scenarios. Each of them used singularly or together, can at a low loss of accuracy improve life of a sensor node by significant amounts. Using a feedback control mechanism as described in the thesis at each sensor node to control resolution and sampling rate, and a delay module to help store data during the period when the logic realizes the need to switch to a higher resolution will allow reduction in power without the loss of accuracy for our purpose. Inexact design in combination with these techniques on the processing side of the network will further improve power consumption of the network with minimal reduction in accuracy. Circuits like inexact adders and multipliers can thus be used for processing logic in wireless sensor networks and the like.

BIBLIOGRAPHY

1. Avinash Lingamneni *et al., "*Designing Energy-Efficient Arithmetic Operators Using Inexact Computing" in *Journal of low power electronics*, 2013

2. Cha Zhang *et al., "*Maximum Likelihood Sound Source Localization and Beamforming for Directional Microphone Arrays" in *IEEE Transactions in Multimedia*, 2007

3. Avinash Lingamneni *et al.,* "Algorithmic Methodologies for Ultra-efficient Inexact Architectures for Sustaining Technology Scaling" in *Proceedings of the 9th conference on Computing Frontiers*, 2012

4. Mart´ı Guerola, Amparo, "Real-Time Sound Source Localization in Videoconferencing Environments", 2013

5. Philippe Bourquin, "Adaptive Sampling for Sensor Networks" in *Seminararbeit Sommersemester*, 2005

6. Meiyan Zhang, Wenyu Cai, "An Energy Effective Frequency based Adaptive Sampling Algorithm for Clustered Wireless Sensor Networks" in *ICCTS*, 2012

7. Frieder Ganz, Payam Barnaghi, Francois Carrez, "Multi-resolution Data Communication in Wireless Sensor Networks", 2014

8. Cha Zhang *et al., "*Why does PHAT work well in lownoise, reverberative environments?" in *ICASSP*, 2008

9. Cesare Allipi *et al.*, "Energy Management in Wireless Sensor Networks with Energy-Hungry Sensors" in *IEEE Instrumentation and Measurement Magazine*, April 2009.

10. Avinash Lingamneni *et al., "*Synthesizing Parsimonious Inexact Circuits through Probabilistic Design Techniques" in *ACM Transactions on Embedded Computing Systems*, May 2013.

11. HC So, YT Chan, "Closed-Form Formulae for Time-Difference-of-Arrival Estimation" in *IEEE Trans Signal Process*, 2008

APPENDIX

Some of the functions used in the simulation and their utility:

createRandomSound() – Creates a random sound at a random time and position using a predefined sound source with a random amplitude within a range. Used to generate the random background noise in the simulation

createRandomLongSignal() – Create a long signal which is treated as an event. Has parameters if what time and length of signal to generate after which it generates a reasonable amplitude random signal with the given parameters.

propogateSoundFromSource() – Propagates generated sound from the sound source to all microphones using the physics defined in the simulation.

Microphone.receiveSound() – ADC catches signal and interrupts processor.

Processor.soundEventDetected() – Processor does the logic processing to decide the sampling rate and resolution of the ADC's in question

The simulation is run using an event queue, which basically is a queue of all the events that happen in order, after creating events randomly during initialization.

Event types:

- CreateSound
- ReceiveSound
- ChangeResolution (Upwards and downwards)
- ChangeSampling (Upwards and downwards)

These events get created internally when initializing the simulation according to parameters and are en-queued in the event queue, which is run at the end to generate simulation results and graphs

Source code for simulation:

```
import math
import random
import matplotlib.pyplot as plt
from random import sample

print "\n"
print "Microphone simulation experiments\n"

timeToChangeMean = 0
highRes = 100
highRate = 100
lowRate = 20
lowRes = 20
sleepRate = 1
sleepRes = 1

class Microphone:
   'Common base class for all Microphones'
   micCount = 0

   def __init__(self, name, posX, posY, rangeOfMic, adc):
      self.name = name
      self.posX = posX
      self.posY = posY
      self.rangeOfMic = rangeOfMic
      self.adc = adc
      Microphone.micCount += 1

   def displayCount(self):
     print "Total Microphones %d" % Microphone.micCount

   def display(self):
      print "Name : ", self.name,  ", posX: ", self.posX, ", posY: ",
self.posY

   def receiveSound(self, event):
      #source = event.source
      #time = event.time
      #amp = event.param1
      eventTime = self.adc.catchSoundSignal(event.time)
      self.adc.interruptProcessor(eventTime, event)


class SoundSource:
   'Common base class for all sound sources'
   sourceCount = 0

   def __init__(self, name, posX, posY, rangeOfSource):
      self.name = name
      self.posX = posX
```

```
        self.posY = posY
        self.rangeOfSource = rangeOfSource
        SoundSource.sourceCount += 1

    def displayCount(self):
      print "Total Sources %d" % SoundSource.sourceCount

    def display(self):
        print "Name : ", self.name,  ", posX: ", self.posX, ", posY: ",
self.posY

    def createSound(self, time, amp):
        print "Sending sound waves to all microphones from", self.name
        self.amp = amp
        # for mic in microphoneArray:
        #    mic.receiveSound()
        #phy.propogateSoundFromSource(self)
        eventQueue.append(Event("createSound", self, self, time, amp, 1))


class Event:

    def __init__(self, eventName, eventSource, eventDestination, eventTime,
eventParam1, priority, eventParam2 = -1):
        self.name = eventName
        self.source = eventSource
        self.destination = eventDestination
        self.time = eventTime
        self.priority = priority
        self.param1 = eventParam1


class ADC:

    def __init__(self, id, samplingRate, resolution, status):
        self.currentRate = samplingRate
        self.defaultRate = samplingRate
        self.status = status
        self.name = id
        self.resolution = resolution
        self.receivedSoundArray = [0] * 110
        self.rate = [1] * 110

    def changeStatus(self, status):
        self.status = status

    def changeRate(self, rate):
        self.currentRate = rate

    def changeResolution(self, resolution):
        self.resolution = resolution

    def reset(self):
        self.currentRate = self.defaultRate
```

```
    def catchSoundSignal(self, receivedTime):
        diff = receivedTime%self.currentRate
        return (receivedTime + self.currentRate - diff)

    def interruptProcessor(self, time, event):
        processor.soundEventDetected(event, time, self)

class Processor:
    'do calculations and simulations in this engine'

    clubbed = []
    counter = 0

    def clubEvents(self, event, adc):
        Processor.counter = Processor.counter + 1
        event.eventParam2 = adc
        Processor.clubbed.append(event)
        if Microphone.micCount == Processor.counter:
            self.findNearestMic(Processor.clubbed)
            self.locateSource(Processor.clubbed)
            Processor.clubbed = []
            Processor.counter = 0

    #def testSource

    def soundEventDetected(self, event, time, adc):
        print "Name : ", adc.name, " - Sound received from ",
event.source.name, "at ",time, "with amp ", event.param1
        print int(time)
        adc.receivedSoundArray[int(time)] = event.param1

        sum = 0
        for i in range(5):
            sum += adc.receivedSoundArray[int(time) - i]
        if (sum/5) > 3:
            #if adc.resolution != highRate:
            eventQueue.append(Event("changeSamplingRate", adc, adc, time +
timeToChangeMean, highRate, 0))
            eventQueue.append(Event("changeResolution", adc, adc, time +
timeToChangeMean, highRes, 0))
            print "high"
        else:
            if (sum/5) < 3 and (sum/5) > 0.2:
                #if adc.resolution == highRate:
                eventQueue.append(Event("changeSamplingRate", adc, adc, time +
timeToChangeMean, lowRate, 0))
                eventQueue.append(Event("changeResolution", adc, adc, time +
timeToChangeMean, lowRes, 0))
                print "low"
            if (sum/5) < 0.2:
                eventQueue.append(Event("changeSamplingRate", adc, adc, time +
timeToChangeMean, sleepRate, 0))
                eventQueue.append(Event("changeResolution", adc, adc, time +
timeToChangeMean, sleepRes, 0))
```

```
            print "sleep"
        self.clubEvents(event, adc)

    def findNearestMic(self, eventsArray):
        print "finding nearest mic to activate"
        eventsArray.sort(key=lambda x: x.param1, reverse=True)
        print 'loudest mic is ', eventsArray[0].eventParam2.name

    def locateSource(self, eventsArray):
        print "locating source from data"

    def createRandomSound(self, time):
        randInt = random.randint(1, len(sourceArray)) - 1
        source = sourceArray[randInt]
        source.posX = random.randint(0, 1000)
        source.posY = random.randint(0, 1000)
        #randTime = random.randint(0, 100)
        source.createSound(time - 0.1, 0)
        source.createSound(time + 0.1, 0)
        source.createSound(time, random.randint(15, 25)*100)
        return randInt

    def createRandomLongSignal(self, time, duration, ranIntChoice):
        ranIntChoice = [x for x in ranIntChoice if x not in range(time, time +
duration)]
        randInt = random.randint(1, len(sourceArray)) - 1
        source = sourceArray[randInt]
        source.posX = random.randint(0, 1000)
        source.posY = random.randint(0, 1000)
        #randTime = random.randint(0, 100)
        source.createSound(time - 0.1, 0)
        source.createSound(time + duration + 0.1, 0)
        for i in range(0, duration*10):
            source.createSound(time + 0.1*i, random.randint(20, 25)*200)
        return (randInt, ranIntChoice)



    def checkForActivity(self, data):
        #if data > mean*1.25:
        #    source.resolution = high
        print 'check for abnormalities to increase resolution and samplingRate'



    #def checkMovingAverage()

class Physics:
    'does the stuff related to physics'

    def __init__(self, name):
        self.name = name

    def propogateSoundFromSource(self, event):

        #print "Propogating sound waves"
```

```
      source = event.source
      for mic in microphoneArray:
         dist = self.findDistance(source.posX, source.posY, mic.posX,
mic.posY)
         if mic.rangeOfMic > dist and source.rangeOfSource > dist:
            eventQueue.append(Event("receiveSound", source, mic, time +
self.calculateTime(dist), event.param1/dist, 0))
            #mic.receiveSound(source, time)

   def findDistance(self, x1, y1, x2, y2):
      return math.sqrt((x1-x2)*(x1-x2) + (y1-y2)*(y1-y2))

   def calculateTime(self, distance):
      return distance/100

   def createSound(self, source):
      source.createSound()

objArray = []
sourceArray = []
microphoneArray = []
adcArray = []

eventQueue = []
completedEvents = []
time = 0

ranIntChoice = range(100)

phy = Physics("PhysicsEngine")
processor = Processor()

"This would create first object of Microphone class"
source1 = SoundSource("Source1", 300, 800, 1000)
source2 = SoundSource("Source2", 700, 900, 1000)
source3 = SoundSource("Source3", 700, 900, 1000)

adc1 = ADC("Adc1", 0.1, 10, 0)
adc2 = ADC("Adc2", 0.1, 10, 0)
adc3 = ADC("Adc3", 0.1, 10, 0)
#adc5 = ADC("Adc5", 0.1, 10, 0)


"This would create first object of Microphone class"
mic1 = Microphone("Mic1", 200, 200, 1000, adc1)
"This would create second object of Microphone class"
mic2 = Microphone("Mic2", 500, 700, 1000, adc2)
"This would create third object of Microphone class"
mic3 = Microphone("Mic3", 800, 200, 1000, adc3)
#mic5 = Microphone("Mic5", 550, 500, 1000, adc5)

objArray.extend((source1, mic1, mic2, mic3))
sourceArray.extend((source1, source2, source3))
microphoneArray.extend((mic1, mic2, mic3))
```

```
adcArray.extend((adc1, adc2, adc3))

Xcoords = []
Ycoords = []
soundsX = []
soundsY = []


for obj in objArray:
   obj.display()

for mic in microphoneArray:
   Xcoords.append(mic.posX)
   Ycoords.append(mic.posY)


print "Total Microphones %d" % Microphone.micCount
print "Total Objects %d" % len(objArray)

#source1.createSound(10, 5)

fig = plt.figure()
ax = fig.add_subplot(111)

x, ranIntChoice = processor.createRandomLongSignal(25, 20, ranIntChoice)
x, ranIntChoice = processor.createRandomLongSignal(75, 15, ranIntChoice)


for i in range(0, 100):
   randtimehere = ranIntChoice[random.randint(0, len(ranIntChoice) - 1)]
   sourceNum = processor.createRandomSound(randtimehere)
   curr = sourceArray[sourceNum]
   if(sourceNum == 0):
      marker = 'bx'
   elif (sourceNum == 1):
      marker = 'gx'
   elif (sourceNum == 2):
      marker = 'yx'


   ax.plot(curr.posX, curr.posY,marker, ms=curr.amp/100)
   #soundsX.append(sourceArray[sourceNum].posX)
   #soundsY.append(sourceArray[sourceNum].posY)



labels = ['mic{0}'.format(i+1) for i in range(len(microphoneArray))]
#plt.plot([1,2,3,4], [1,4,9,16], 'ro')
ax.plot(Xcoords, Ycoords, 'ro')
for label, x, y in zip(labels, Xcoords, Ycoords):
    ax.annotate(label, xy=(x,y), xytext = (-5, 5),  textcoords='offset
points')
```

```
#plt.plot([1,2,3,4], [1,4,9,16], 'ro')
ax.plot(Xcoords, Ycoords, 'ro')
for xy in zip(Xcoords, Ycoords):
# <--
    ax.annotate('(%s, %s)' % xy, xy=xy, xytext = (-20, 20),
textcoords='offset points') # <--
#plt.plot(soundsX, soundsY,'bx', ms=10)
ax.axis([0, 1000, 0, 1000])
#ax.show()

fig1 = plt.figure()
ax1 = fig1.add_subplot(111)

ax1.axis([0, 100, 0, 100])


timeArray = []
ampArray = []

#Simulation loop

eventQueue.sort(key=lambda x: x.time)

while(len(eventQueue)!= 0):

   currentEvent = eventQueue.pop(0)
   time  = currentEvent.time
   if currentEvent.name == "createSound":
      phy.propogateSoundFromSource(currentEvent)
      ax1.plot(currentEvent.time, currentEvent.param1/100,'ro', ms = 2)
      timeArray.append(currentEvent.time)
      ampArray.append(currentEvent.param1/100)
   elif currentEvent.name == "receiveSound":
      currentEvent.destination.receiveSound(currentEvent)
      completedEvents.append(currentEvent)
   elif currentEvent.name == "changeResolution":
      currentEvent.source.changeResolution(currentEvent.param1)
      currentEvent.source.rate[int(currentEvent.time)] = currentEvent.param1
      completedEvents.append(currentEvent)
   elif currentEvent.name == "changeSamplingRate":
      currentEvent.source.changeRate(currentEvent.param1)
      completedEvents.append(currentEvent)

fig3 = plt.figure()
ax3 = fig3.add_subplot(111)
ax3.plot(adcArray[0].receivedSoundArray)
fig4 = plt.figure()
ax4 = fig4.add_subplot(111)
ax4.plot(adcArray[1].receivedSoundArray)
fig5 = plt.figure()
ax5 = fig5.add_subplot(111)
ax5.plot(adcArray[2].receivedSoundArray)

print 'rate'
print adcArray[0].rate
```

```
fig6 = plt.figure()
ax6 = fig6.add_subplot(111)
ax6.plot(adcArray[0].rate)
ax6.axis([0, 150, 0, 200])
fig7 = plt.figure()
ax7 = fig7.add_subplot(111)
ax7.plot(adcArray[1].rate)
ax7.axis([0, 150, 0, 200])
fig8 = plt.figure()
ax8 = fig8.add_subplot(111)
ax8.plot(adcArray[2].rate)
ax8.axis([0, 150, 0, 200])


ax1.plot(timeArray, ampArray)
plt.show()
#print sample(xrange(100), 20)
#print ranIntChoice

print "Power calculations"
powerOld = [0]*len(adcArray)
powerNew = [0]*len(adcArray)

for j in range(len(adcArray)):
   for i in range(len(adcArray[0].rate)):
      #print str(i) + " " + str(j)
      powerNew[j] += adcArray[j].rate[i]
      powerOld[j] += 100
for x in range(len(adcArray)):
   print 'Power ADC ' + str(x) + ' Old ' + str(powerOld[x])
   print 'Power ADC ' + str(x) + ' New ' + str(powerNew[x])
```