Fall 2013

Embedded Cloud System for Ann-Cod Analysis Using UV Spectroscopy

Kaushik Patra
San Jose State University

EMBEDDED CLOUD SYSTEM FOR ANN-COD ANALYSIS USING UV SPECTROSCOPY

A Thesis

Presented to

The Faculty of the Department of Computer Engineering

San Jośe State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Kaushik Patra

December 2013

The Designated Thesis Committee Approves the Thesis Titled

EMBEDDED CLOUD SYSTEM FOR ANN-COD ANALYSIS USING UV SPECTROSCOPY

by

Kaushik Patra

APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING

SAN JOŚE STATE UNIVERSITY

December 2013

Dr. Hua Harry Li	Department of Computer Engineering
Dr. Donald Hung	Department of Computer Engineering
Dr. Robert H. Morelos-Zaragoza	Department of Electrical Engineering

ABSTRACT

EMBEDDED CLOUD SYSTEM FOR ANN-COD ANALYSIS USING UV SPECTROSCOPY

by Kaushik Patra

One of the many parameters indicating water quality is chemical oxygen demand (COD), which is an indirect measurement of the amount of organic compound material in water. There have been many studies, in both academia and the industry, to analyze the COD content of water using spectral analysis. The proposal of this thesis was to study, analyze, and identify methods to determine the presence of COD using UV spectroscopy data and an artificial neural network (ANN) in a cloud-connected embedded system. The system was implemented using an ARM11 board and a portable spectrometer. Light in the UV range was used to analyze the water sample. As an analysis strategy, the spectral data were converted into a real number value in the range of 0 to 1. Twenty equidistance samples were taken out of the converted data to be fed into the ANN, and the ANN was trained with known samples to identify any presence of COD. Experiments used laboratory-calibrated water samples with known COD and some real-life water samples. All the experiments showed that the implemented system could successfully indicate the presence or absence of COD in the given water sample. The system also successfully demonstrated the application of a cloud-connected embedded system to an area in environmental engineering. This indicated that the system was a bridge between computer and environmental engineering.

ACKNOWLEDGEMENTS

First, I would like to express my humble and sincere gratitude to my master's thesis guide, Dr. Hua Harry Li, for his continuous support and encouragement in this work. I owe a lot to him for his active involvement and stream of ideas, which enriched this work in every aspect.

Besides my thesis advisor, I would also like to thank the rest of the thesis committee: Dr. Donald Hung and Dr. Robert H. Morelos-Zaragoza, who provided their valuable and insightful comments and constructive feedback on this thesis work.

I would also like to extend my thanks to Arch Shaw, alumni of the Computer Engineering Department, whose master's project, *Cloud Computing Enabled Portable Smart Spectrophotometer*, was a cornerstone for this thesis.

My sincere thanks also go to my loving wife, Purna Chatterjee Patra, who took care of all of the family chores to provide me adequate time for thesis work in addition to my daily full-time office job. She also provided her valuable feedback on the thesis manuscript and often acted as an extra pair of eyes to weed out technical and compositional errors in my writing.

Last but certainly not least, my heartfelt thanks go to my beloved daughter, Debasmita Patra, who despite being only a toddler gives a sense of importance to my thesis. She has always encouraged me by repeatedly reminding me that there will be plenty of playtimes with her once my "school assignment" is done.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION TO THE THESIS TOPIC

1.1 Introduction

Water is the most important factor in sustaining human civilization. Water not only provided the first breeding ground for life but also played a pivotal role in supporting life on earth. Water has allowed the human species to evolve and secure a unique evolutionary position that is unparalleled by any other species. The ubiquitous influence of water on human life, as well as on civilization, is unquestionable. The uses of water are diverse and include direct human consumption, agriculture, industry, and scientific studies. As a universal solvent, water can dissolve many inorganic and organic materials. As a result, obtaining water without any contamination is difficult. However, for many of the common uses of water, including human consumption, it is neither appropriate nor practical to use pure water. Water standards, including acceptable concentrations of different inorganic and organic materials dissolved in water, have been suggested for different usage scenarios. These suggestions indicate the overall water quality for a particular type of usage. Many standard parameters have been developed to indicate water quality. The two best known parameters are chemical oxygen demand (COD) [1] and biological oxygen demand (BOD) [2].

This thesis aimed to explore methodologies for determining the COD of water and develop an economical methodology for COD determination using a cloud-enhanced generic embedded system. The proposed system used an artificial neural network

(ANN)-based analysis of spectroscopy data obtained from a low-end spectrometer. The spectroscopy data were collected using a source that produced light of 190 to 885 nm (i.e., UV and visual light).

This section has three subsections. Section 1.2 includes a brief discussion of what water treatment processing is. It also explains why water treatment is necessary. A brief discussion of water quality and standards has also been included in the background discussion. An introduction to the COD parameters for water quality determination is also included in this section. Section 1.3 describes the contemporary problems and issues in water quality measurement and monitoring. Section 1.4 describes the overall objective of the study and experiment presented in this thesis.

1.2 Background

Although 70% of the earth's surface is covered with water, more than 96% of the water is contained in the ocean [3]. Ocean water is not useful for most daily human activities. The remaining water is contained underground and in lakes, rivers, swamps, and the atmosphere. Table 1 shows the global distribution of water. This table also accounts for the water content of living organisms (known as "biological water").

Table 1. One Estimate of Global Water Distribution

Water source	Water volume, in cubic miles	Water volume, in cubic kilometers	Percent of fresh-water	Percent of total water
Oceans, seas, and bays	321,000,000	1,338,000,000	--	96.54
Ice caps, glaciers, and permanent Snow	5,773,000	24,064,000	68.6	1.74
Groundwater	5,614,000	23,400,000	--	1.69
Fresh	2,526,000	10,530,000	30.1	0.76
Saline	3,088,000	12,870,000	--	0.93
Soil moisture	3,959	16,500	0.05	0.001
Ground ice and permafrost	71,970	300,000	0.86	0.022
Lakes	42,320	176,400	--	0.013
Fresh	21,830	91,000	0.26	0.007
Saline	20,490	85,400	--	0.007
Atmosphere	3,095	12,900	0.04	0.001
Swamp water	2,752	11,470	0.03	0.0008
Rivers	509	2,120	0.006	0.0002
Biological water	269	1,120	0.003	0.0001

Source: Igor Shiklomanov, "World Fresh Water Resource," *Water in Crisis: A Guide to the World's Fresh Water Resources*, ed. Peter H. Gleick (Oxford University Press, New York, 1993).

Freshwater is used much more often than saline water. Fig. 1 shows some sample usage ratio statistics for groundwater versus freshwater and saline water versus freshwater. The statistics clearly show that the usage of freshwater is much more prominent than the usage of saline water. Although the natural water cycle [4] maintains the supply of freshwater, water shortages are a growing issue in different parts of the world. This water crisis affects not only the human race but also every living being on the planet. There are many manifestations of the water crisis, including the following:

1. About 884 million people worldwide do not have access to safe drinking water [5].
2. About 2.5 billion people worldwide do not have access to water for sanitation and waste disposal [6].
3. Overuse of groundwater causes low agricultural yield [7].
4. Pollution of water resources affects biodiversity [8].

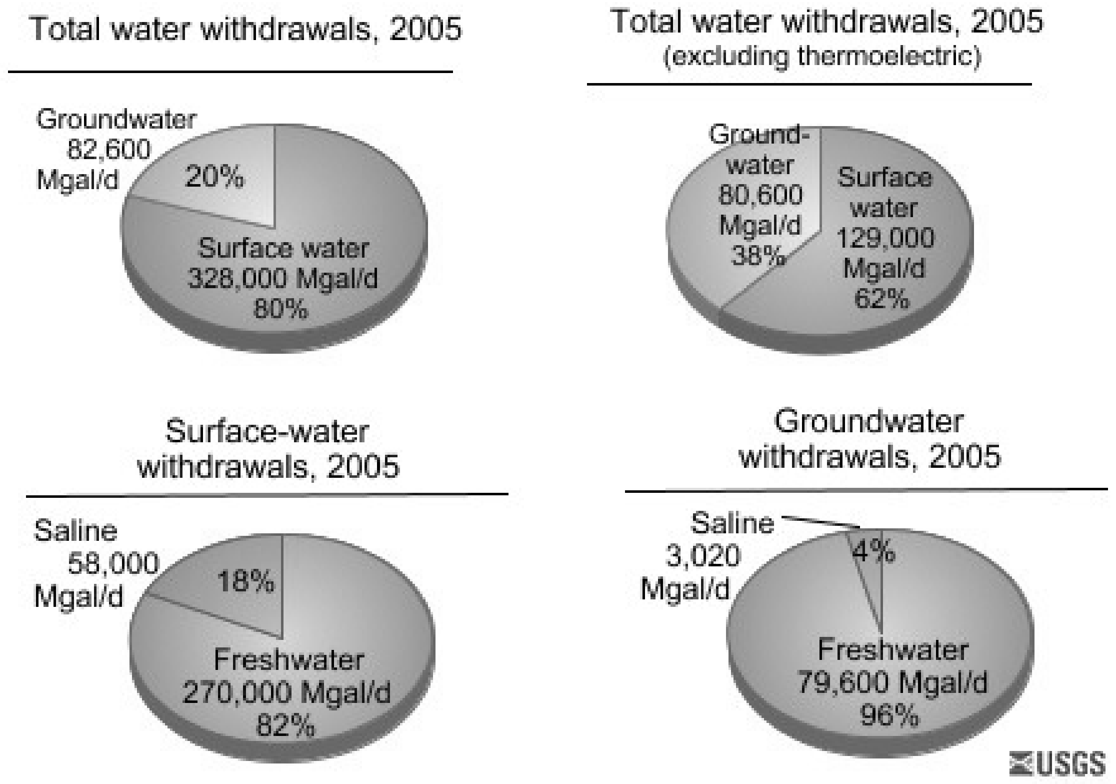


Fig. 1. Sample Water Usage Statistics

Source: USGS, *How much water is there in, on and above the Earth?* http://ga.water.usgs.gov/edu/earthhowmuch.html, (Accessed May 20, 2012).

There are two plausible solutions to this growing problem of water crises.

1. Recycle water within a system (e.g., recycle of sanitation water within a large building).
2. Remove contaminants from water before returning the water from a system to nature (e.g., treat water waste from industry before releasing it to rivers or the ocean).

Both of these strategies require a process for eliminating contaminants from water. In general, this process of removing contaminants from water is known as the water-treatment process. A water-treatment procedure may involve physical, chemical, and even biological processes with the goal of removing contamination from water so that the treated water may be reused for a specific purpose like drinking, sanitation, or medical, agricultural, or industrial usage [9]. Each type of water usage has its own standards and recommendations for the upper limit of contaminants. For example, water used for medical purposes should have much less contamination than common drinking water, if any at all. Even industrial usage of water varies from industry to industry [10]. An example of drinking water standards can be found in the web documentation of the U.S. Environmental Protection Agency [11]. This list subdivides contaminants into six categories: microorganisms, disinfectants, disinfection byproducts, inorganic chemicals, organic chemicals, and radionuclides.

One of the challenges for any water treatment plant is to analyze the water quality and the standard of the treated water before releasing it to nature for reuse or to recycle.

COD is one such measurement of quality and indicates the amount of organic compounds in water [1]. In acidic conditions, nearly every organic chemical can be fully oxidized to carbon dioxide (CO_2) and ammonia (NH_3) using a strong oxidizing agent [1]. Equation 1 is a simple formula to compute the amount of oxygen needed for complete oxidization of an organic compound [1].

$$C_nH_aO_bN_c + \left(n + \frac{a}{4} - \frac{b}{2} - \frac{3c}{4}\right)O_2 \rightarrow nCO_2 + \left(\frac{a}{2} - \frac{3c}{2}\right)H_2O + cNH_3 \qquad \text{Equation 1}$$

COD measurement indicates the amount of oxygen needed to oxidize organic compound contamination in water. Thus, COD can indicate water quality in terms of organic compound contamination.

1.3 Problem Statement

Contemporary COD analysis methods involve either one of the pure chemical, electrochemical, spectrometry/colorimetry-based, or photo-catalytic-based instruments. According to a publication from the University of Georgia,

> The most popular current testing method for COD involves using sealed and heated (i.e., closed reflux) low-range (3−150 ppm) or high-range (20−1,500 ppm) pre-prepared vials that change color from orange to green based on the amount of oxidation and that are read using a laboratory colorimeter. [12]

A person must conduct the COD analysis procedure because each of the methods mentioned above is chemical reagent based. The steps are quite involved and complex including hazardous closed reflux processes in an acidic test environment. The contemporary methods of COD measurement usually involve a setup process [13] and environmentally unfriendly reagents (e.g., mercury) [16]. Therefore, disposal of wastes from the contemporary COD determination experiment create additional overhead. Moreover, because these methods are a chemical oxidation-based measurement procedure, they cannot account for certain organic chemicals, such as aromatic hydrocarbons, volatile organic compounds, pyridine and related compounds, ammonia, and straight chain aliphatic [13, 14, 16]. In addition, contemporary COD analysis procedures are sensitive to the experimental environment. None employs automatic adaptation of experimental environment variability.

Except for the photo-catalytic process [15] (which takes 2–3 minutes), other popular processes need 2–3 hours to be conducted [12, 16]. All the instruments involved (i.e., colorimeter or PeCOD^{TM}) are stand-alone devices and custom made for a specific set of analyses. The general extensibility of functionality to determine new types of chemicals and compounds are limited in such custom-made systems. In addition to the basic cost of a customized machine, all these methods of chemical reagent-based analysis have recurring costs associated with their practical applications. Hence, over time, the total cost of water quality analysis becomes significant.

1.4 Objectives of the Study

The main objective of this study was to find a cheaper, faster, mobile, extendible, and reliable solution for online COD analysis. Therefore, a complete study of contemporary COD analysis methods was undertaken, and the disadvantages of these methods were identified. To reduce the recurring charges on chemical reagents, a method of COD analysis without any chemical processes must be implemented. To reduce base costs of purchasing special-purpose machines for COD analysis, a method using a general-purpose system must be considered. Such a method would also create a possibility for extending the analysis system to a different set of parameters other than COD. The system also must be portable for in situ COD analysis. Because analysis can produce a huge amount of data, a system that can handle and analyze large amounts of data must be considered. Compensation for analysis condition variability must also be taken into consideration.

The basic idea of this thesis was to study and experiment with the possibility of using a general-purpose mobile computing system integrated with a cloud [17] and using a relatively cheap spectroscopy analysis instrument working in the UV-Vis frequency range. An ANN [18] was used to implement an adaptable system in a variable analysis environment. Usage of a general-purpose mobile computing platform gave the advantages of portability and reduction of the base cost to engineer such a system compared to a special-purpose machine. The success of this experiment should prove that a COD analysis application could be built for contemporary smartphones. In that case, COD analysts only need to carry their smartphones and a lightweight spectroscopy device on-site. The proposed system, which is built on top of a general-purpose computing system, has the advantage of functional extendibility. New tests can be added to the software applications to accommodate more types of tests and analyses. By integrating the system with a cloud [19], analysis can be performed on a remote server. This would bring two advantages—along with the obvious advantage of the ability to execute complex and powerful algorithm for data analysis. One advantage would be not overloading the mobile system. Therefore, the system can focus on data gathering and communication to the cloud the server only. The other advantage would be a further reduction of costs by using a pay-per-use model [19].

CHAPTER TWO

LITERATURE SURVEY ON COD MEASUREMENT TECHNIQUE

2.1 General COD Measurement Techniques

Three major techniques for general COD measurement were identified in the literature survey. All three techniques involve chemical processes. The first technique is purely based on chemical processes involving measurement of the amount of reagent used. The next two processes are a combination of chemical and spectrometry or photo-catalytic analysis.

2.1.1 Chemical Reflux and Titration-Based Method

Zhang's *Fundamental of Environmental Sampling and Analysis* [16] has a clear description of the process of COD measurement using chemical reflux and the titration-based method. Fundamentally, in this process, the water sample is refluxed with potassium dichromate ($K_2Cr_2O_7$) in a strong acid condition. Often the acidic condition is created by mixing concentrated sulfuric acid (H_2SO_4) into the water sample. The reflux process is conducted for about 2 hours. During this process, the organic compound mixed with sample water is converted into carbon dioxide (CO_2). Any organic nitrogen is converted into ammonium ions (NH_4^+), which further can be converted into nitrate compounds. In this reflux process, the amount of potassium dichromate digested depends on the amount of organic compound present. Therefore, by measuring the amount of potassium dichromate digested in this process, COD can be measured indirectly.

To detect how much potassium dichromate is left after the reflux process, a chemical titration process is performed using ammonium iron (II) sulfate [$(NH_4)_2Fe(SO_4)_2.6H_2O$], which is also known as ferrous ammonium sulfate (FAS) or Mohr's salt [20]. This salt is classified as double salt of ferrous sulfate and ammonium sulfate [20]. In this titration process, the chelating agent 1,10-phenanthroline (ferroin, $C_{12}H_8N_2$) is used as the indicator. At the start of the titration process, the refluxed water sample color is between yellow and orange brown depending on the concentration of undigested potassium dichromate (the $Cr_2O_7^{2-}$ is responsible for this color). Thereafter, the color changes to a blue-green at the start of the titration process, which is the color of the Cr_3^+. The titration process is concluded when the solution sharply changes to reddish brown, which is the color of $Fe\{C_{12}H_8N_2\}$. By measuring the amount of FAS used for titrating, the excess amount of potassium dichromate and thus the COD amount in the sample water can be determined. The detailed process and formula can be found in the reference book mentioned at the beginning of this subsection.

2.1.2 Colorimeter-Based Method

The colorimeter-based method is similar to the chemical reflux-based method described in section 2.1.1. The main difference is that instead of measuring the concentration of undigested potassium dichromate using a titration method, the colorimeter-based method uses colorimeter [23] to determine the concentration of undigested potassium dichromate. A colorimeter uses principles of colorimetry, which is a technique "used to determine the concentration of colored compounds in solution" [21]. A colorimeter uses references to a known concentration of a substance under examination in a solution to compute the concentration of the substance in that target solution. A detailed description of the colorimeter-based method can be found in the University of Georgia publication *Understanding Laboratory Waste Water Test: I. ORGANICS (BOD, COD, TOC, O&G)* [12]. According to this method, many samples of known COD (usually calibrated with KHP or potassium hydrogen phthalate solution) are used as a reference to a colorimeter. Because this colorimeter-based technique is dependent on the color absorbance of the material, it is important to select the right color filter in the colorimeter. The colorimeter should use light of such a color that the sample can absorb. For example, red light can be used to detect the concentration of a blue-colored solution [25]. Fig. 2 and Fig. 3 are pictures of a very simple colorimeter from Vernier Software and Technology (http://www.vernier.com/company/).

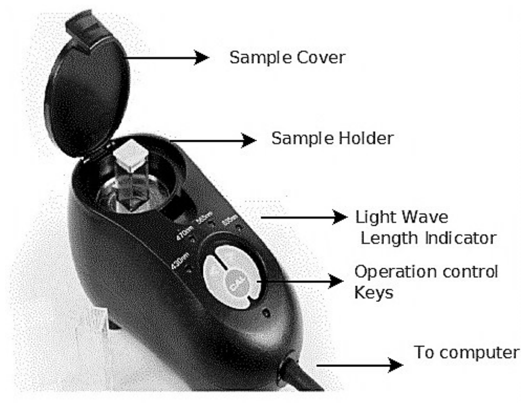
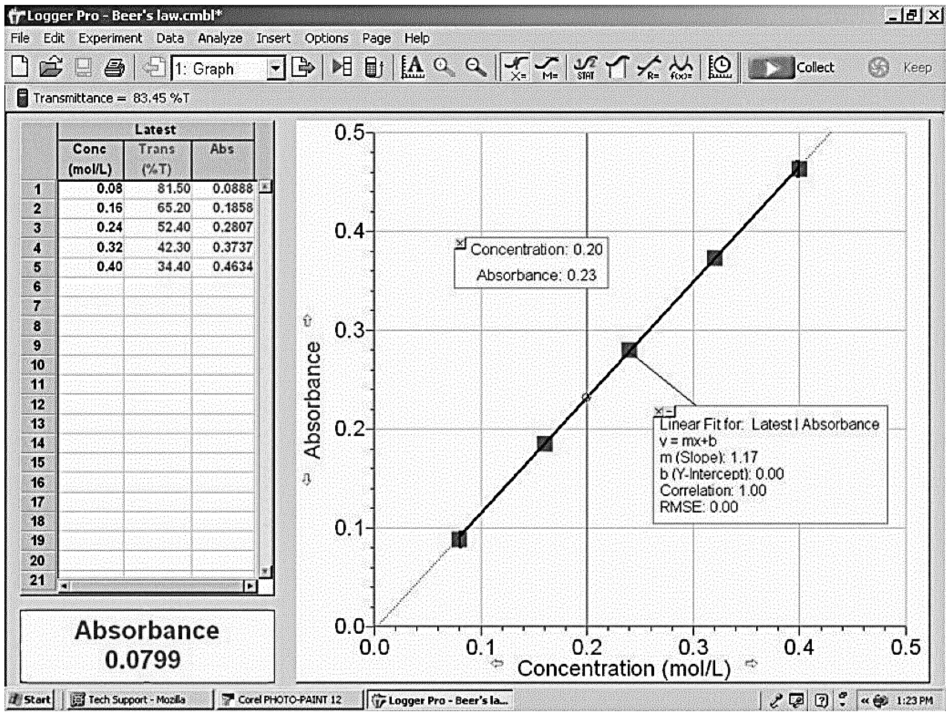


Fig. 2. A Simple Colorimeter

[Composed from the original source]

Source: Vernier Software and Technology, *Colorimeter*, http://www.vernier.com/products/sensors/col-bta/ (Accessed June 15, 2012).

This simple colorimeter offers four different ranges of color wavelength (430, 470, 565, and 635 nm). The corresponding software captures the absorbance of the light through the sample. The sample is put into the sample holder and covered with a sample cover prior to the start of the operation. Using this colorimeter and software interface, calibrated samples can be used to build the absorbance curve with respect to the sample concentration. Once the absorbance curve is determined, an unknown sample can be used to measure concentration through the mathematical method of curve fitting, which is implemented using the associated software.



Transmittance = 83.45 %T

	Latest		
	Conc (mol/L)	Trans (%T)	Abs
1	0.08	81.50	0.0888
2	0.16	65.20	0.1858
3	0.24	52.40	0.2807
4	0.32	42.30	0.3737
5	0.40	34.40	0.4634
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
19			
20			
21			

Absorbance
0.0799

Concentration: 0.20
Absorbance: 0.23

Linear Fit for: Latest | Absorbance
y = mx+b
m (Slope): 1.17
b (Y-Intercept): 0.00
Correlation: 1.00
RMSE: 0.00

Fig. 3. Software for the Colorimeter Operation

Source: Vernier Software and Technology, *Colorimeter*, http://www.vernier.com/products/sensors/col-bta/ (Accessed June 15, 2012).

2.1.3 Photo-Catalytic Method

The ManSci Inc. (http://www.mansci.com/) developed patented technology based on the photo-catalytic method. This technology uses a sensor with UV-activated nanoparticulate titanium dioxide (TiO2) [15]. External electric circuitry measures the amount of current produced by this sensor. Fig. 4 shows how the photo-catalytic sensor works.

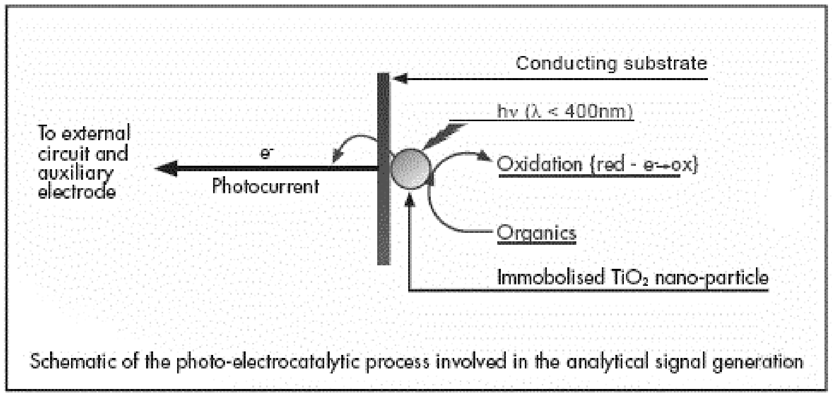

Conducting substrate
hν (λ < 400nm)
To external circuit and auxiliary electrode
e⁻
Photocurrent
Oxidation (red - e⁻→ox)
Organics
Immobolised TiO₂ nano-particle
Schematic of the photo-electrocatalytic process involved in the analytical signal generation

Fig. 4. The PeCOD Machine and the Work Mechanism.

[Composed from the following source]

Source: ManSci Incorporation, *COD Analysis*,

http://www.mansci.com/products/PeCOD/index.html#WhatisPeCOD (Accessed June 15, 2012).

UV radiation creates a photo hole on the TiO2 sensor. It also has very high oxidizing power that guarantees complete oxidation of organic compounds in the sample. The electric circuit measures the number of electrons released, which is proportional to the amount of organic material oxidized in the process. The graph in Fig. 5, published by the same company, shows the effectiveness of the PeCOD instrument.

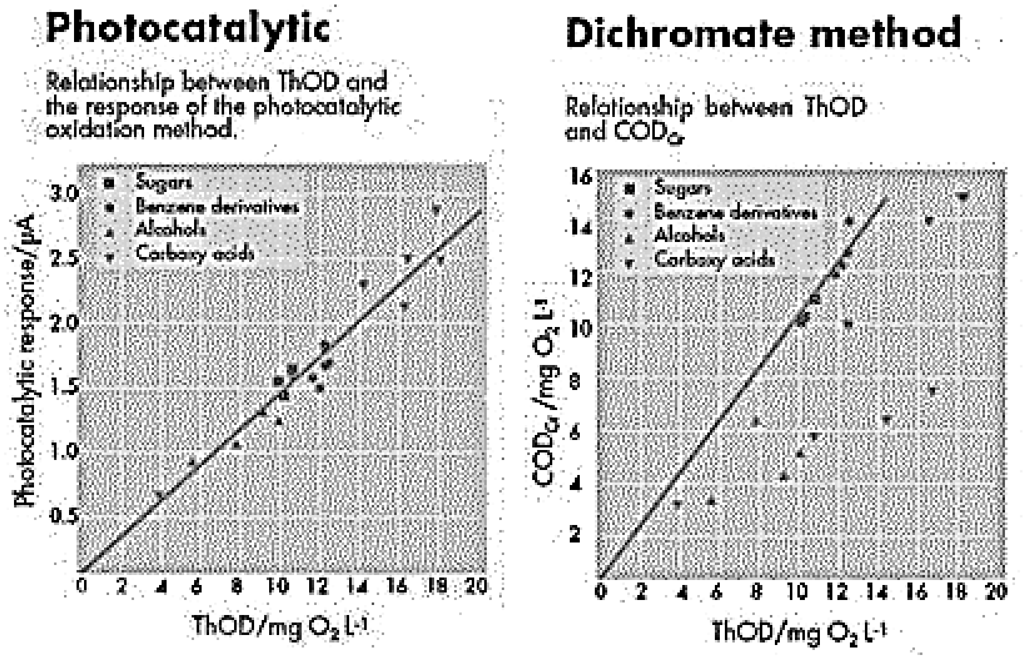


Fig. 5. Photo-Catalytic versus Dichromate Method

[Composed from the following source]

Source: ManSci Inc., *COD Analysis*, http://www.mansci.com/products/PeCOD/index.html#WhatisPeCOD

Fig. 5 shows the theoretical COD on the x-axis and measured COD on the y-axis. With this data, the photo-catalytic method is proven to have closer correlation with theoretical COD values than the conventional dichromate method.

2.2 UV-Vis Spectroscopy for COD Detection

Although there is a plethora of methodologies proposed for fast COD determination, the most popular methodology is based on an indirect spectroscopic technique due to its simplicity and fast response time [28]. This section not only describes such techniques but also explains the basic concept of spectrometry physics, which should help to explain the proposed section topic. Most of the contents of this subsection have been derived from the chapter "The Basics of Spectrophotometry Measurement" from by Thomas and Burgess's *UV-Visible Spectrophotometry of Water and Wastewater* [26].

2.2.1 Physics of Spectroscopy

The physics of spectroscopy has been studied for centuries by many eminent scientists including Sir Isaac Newton, Albert Einstein, Niels Bohr, and Max Planck. Light, or in general electromagnetic radiation (EMR), takes a central role in the process of spectroscopy. EMR interacts with atoms and molecules in a specific way. A part of EMR is absorbed within a material. This phenomenon produces a characteristic absorption profile (or emission profile) for the material under investigation. The process of spectroscopy relies heavily on this interaction.

The ability to detect emission profiles for EMR is determined by the range of visible colors. The property of EMR that determines color is wavelength. Usually EMR is represented as a sine curve and the wavelength is defined as the peak-to-peak distance between two adjacent peaks of the sine curve as shown in Fig. 6.

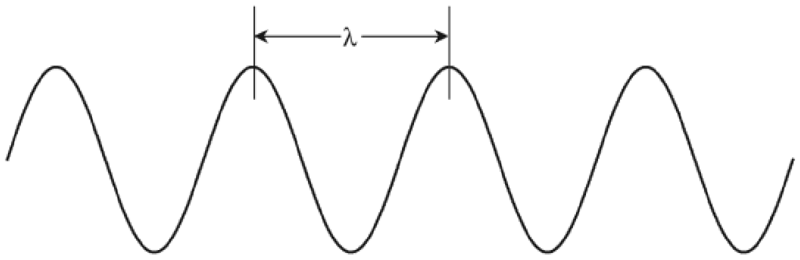


Fig. 6. Wavelength for an EMR

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

The mathematical symbol for wavelength is λ. The other physical characteristic of the EMR is its frequency (ν), which is expressed in Equation 2, where *c* is the speed of light.

$$\nu = \frac{c}{\lambda} \qquad \text{Equation 2}$$

There is an infinite range of wavelengths for EMR, although only a tiny range is detectable by the human eye, which is known as visible light. Fig. 7 shows the ranges of the EMR spectrum, which include the range of visible light.

Notably, the human eye does not have the same response for all visible EMR wavelengths. The response from a human eye depends on its receptor cells. Different ranges of EMR wavelength simulate the eye’s receptor cells in different amounts [27]. Fig. 8 shows the human eye response pattern for visible EMR.

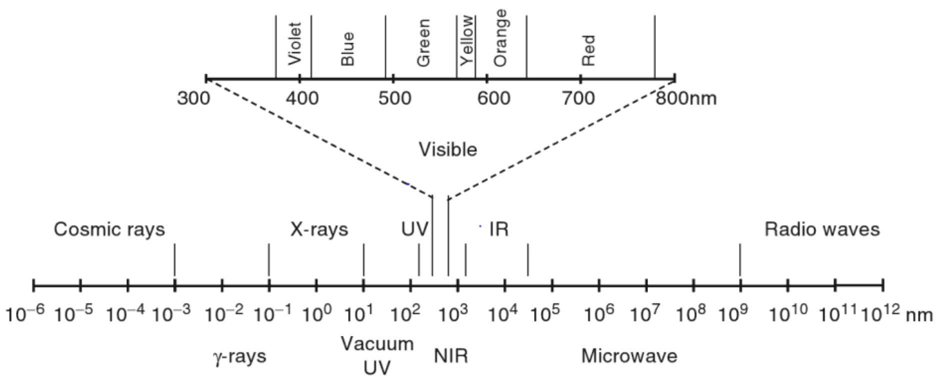

Violet
Blue
Green
Yellow
Orange
Red
300
400
500
600
700
800nm
Visible
Cosmic rays
X-rays
UV
IR
Radio waves
10^{-6} 10^{-5} 10^{-4} 10^{-3} 10^{-2} 10^{-1} 10^{0} 10^{1} 10^{2} 10^{3} 10^{4} 10^{5} 10^{6} 10^{7} 10^{8} 10^{9} 10^{10} 10^{11} 10^{12} nm
γ-rays
Vacuum UV
NIR
Microwave

Fig. 7. The EMR Spectrum

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

A special electronic semiconductor device is used for all practical spectrometry applications. This device is known as a charge-coupled device or CCD [28]. This special device is able to detect a wide range of EMR wavelengths. It is used not only as a spectroscopic sensor but also in digital image capturing and other imaging areas. These sensors convert incoming light particles or "photons" into digitally accounted numbers for further processing by digital data processors.

Light, or EMR, presents dual behavior as a wave and a particle. The particles are known as photons. Equation 3 shows the amount of energy E carried by a photon particle of wavelength λ. This equation contains two constants: *h* is Planck's constant (6.63×10^{-34} Js) and *c* is the speed of light in a vacuum (2.998×10^{8} m/s).

$$E = \frac{hc}{\lambda} \times 10^9 \qquad \text{Equation 3}$$

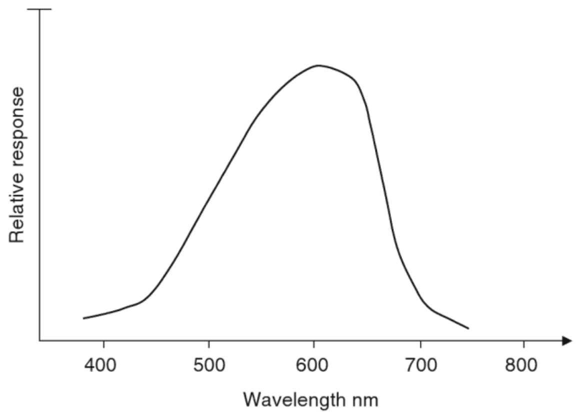


Fig. 8. Relative Response of the Human Eye to Visible EMR

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

When an energy beam falls into a material's electron cloud, the phenomenon of discrete and quantized absorption of energy occurs. In a very simplified sense, this quantized absorption promotes an electron of the material from its ground state to its excited state. Depending on the amount of energy required to bring an electron from its ground state to its excited state, the amount of absorbed EMR of a specific wavelength can be computed using Equation 3. Because different materials exhibit a need for different amounts of quantized energy for the orbital promotion of an electron, different materials exhibit different spectra of absorption. Additionally, the amount of EMR absorbed by a material also depends on the amount of material used in the analysis.

Hence, the absorption characteristics of the material can be used to detect the material, as well as determine the amount of material. Fig. 9 shows a simplistic model and a calculation of the electron promotion and EMR absorption relationship.

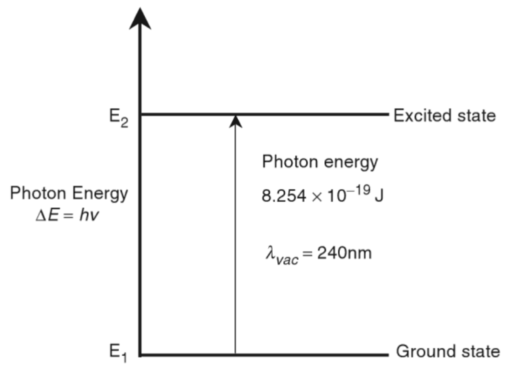


Fig. 9. Photon Capture by Molecule

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

In addition to absorption, the phenomena of transmission, reflection, refraction, scattering, luminescence, and chiro-optical phenomena also occur when an EMR falls onto a material. Any good spectroscopic process aims to minimize the effects of these additional phenomena.

Because electron transition occurs at a high energy level, absorption occurs in the UV region rather than in the infrared or microwave region. Even a simple molecule like benzene presents a complex spectroscopic pattern as in Fig. 10.

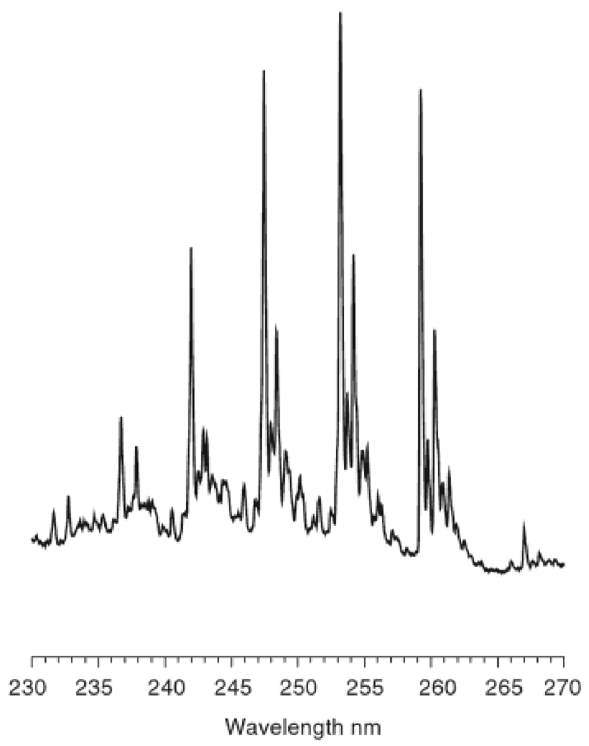


Fig. 10. Vapor State Spectrum for Benzene

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

Different parts of a molecule characteristically absorb different parts of the EMR spectrum. These active parts are known as chromophores. The wavelength a chromophore absorbs depends on the constitution of the chromophores. As a result, different chromophores present in a molecule and their quantity create unique spectral absorption signature for a specific molecule.

The absorption of a specific wavelength depends on the type of transition of the electron. There are three types of molecular orbits in the ground state: sigma (σ), pi (π)

and nonbonding (*n*). There are two types of molecular orbits in the excited state: sigma star (σ^*) antibonding and pi star (π^*) antibonding. There are four types of transitions from the ground-state molecular orbit to the excited-state molecular orbit as in Fig. 10.

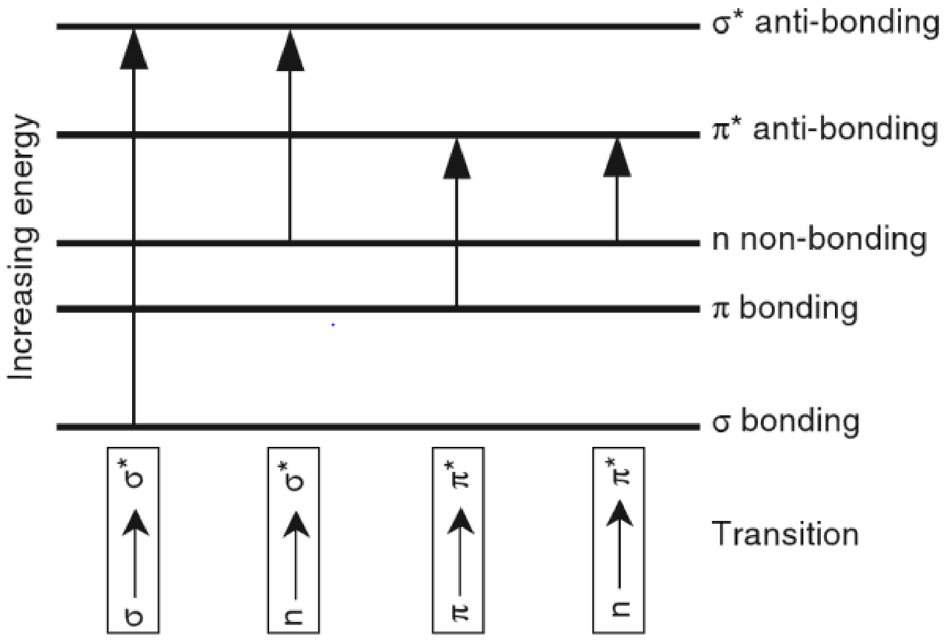


Fig. 11. Molecular Orbit Transitions

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

Different chromophores cause different orbital transitions for electrons. As a result, they absorb EMR of different wavelengths. Fig. 12 shows different absorption spectra per building block of organic compounds.

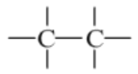
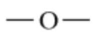
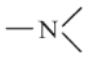
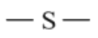
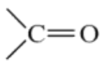
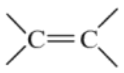
Chromophore	Transition	Approximate wavelength of maximum absorption (nm)
	$\sigma \rightarrow \sigma^*$	150
	$n \rightarrow \sigma^*$	185
	$n \rightarrow \sigma^*$	195
	$n \rightarrow \sigma^*$	195
	$\pi \rightarrow \pi^*$ $n \rightarrow \pi^*$	170 300
	$\pi \rightarrow \pi^*$	170

Fig. 12. Table of Absorption Maxima for Isolated Chromophores

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

After radiation goes through a solution of a target material into a nonpolar solvent like water, there are two fundamental laws and one composite law to describe the relation between the intensity of emerging EMR and the intensity of attenuated EMR. Let I_0 be the intensity of the parallel beam EMR of wavelength λ falling onto a solution of thickness *b*. Let *M* be the amount of concentration of the EMR absorber in the solvent. If *I* is the attenuated intensity of the EMR parallel beam, then Lambert's law gives the following Equation 4. The term k_λ in this equation is a wavelength-specific constant.

$$dI = -k_\lambda db$$ Equation 4

In this same scenario, Beer's law gives the following Equation 5.

$$dI = -k_\lambda dM \qquad \text{Equation 5}$$

Combining Lambert and Beer's law, the following Equation 6 can be obtained as the composite analytical formulation, which is also known as Beer-Lambert law.

$$log_{10}\frac{I_0}{I} = \frac{k'_\lambda bM}{2.303} \qquad \text{Equation 6}$$

This composite constant k'_λ / 2.303 is known as molar absorptivity, and the left side of Equation 6 is known as the absorbance, often symbolized as A.

2.2.2 Spectroscopy in COD Analysis

There are many different types of organic material in wastewaters. Each absorbs part of the EMR in the UV-Vis range according to its characteristics. Hence, the resultant spectroscopic data are compound effects of all the participating materials. Because there can be a high number of different organic materials present in wastewater, it is very difficult to analyze a specific organic compound, except if one such specific compound presents in very high concentrations as in industry wastewater. "This is the reason why the measurement of aggregate organic parameters (oxygen demand, for example) and constituents (phenol index, for example) is most often used for waste water quality control" according to Thomas and Theraulaz [30].

There are two main approaches to determining aggregated parameters for wastewater from the UV-Vis spectroscopic data. Fig. 13 presents the top-level idea for both approaches. The first approach is to determine one or two wavelengths that present a useful correlation to the parameter value to be determined. The second approach is to use multiwavelength exploitation techniques. Historical information shows that the multiwavelength exploitation is more useful than the other approach in aggregated parameter determination [30].

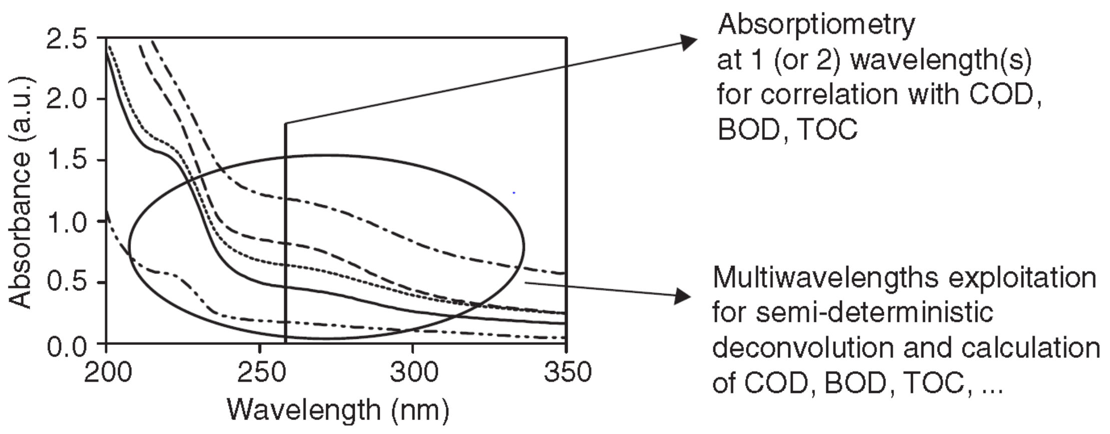


Fig. 13. Main Approach for the Aggregated Parameter Determination

Source: Olivier Thomas and Christopher Burgess, *UV-visible Spectrophotometry of Water and Wastewater, Volume 27 (Techniques and Instrumentation in Analytical Chemistry)*

Many scholarly articles [31-45] dealt with UV-Vis spectroscopy data analysis. Most multiple wavelength analyses use multicomponent analysis techniques, like multiple linear regression [41], partial least squares regression [42], or ANNs [37, 43-45]. All of these approaches have been researched to minimize the determination time of the aggregated parameter, such as COD, needed to analyze the UV-Vis spectral absorption characteristics of a given water sample. The target for devising such a procedure is also to maintain a level of accuracy close to that of chemical reagent-based procedures.

2.3 ANN-Based Approach

The ANN study and its engineering application have continued for decades. McCullouch and Pitts [46] proposed the first mathematical model for a neural network in 1943. Later, Rosenblatt [47] developed a perception model to implement a pattern recognition algorithm based on a two-layer learning computer network. However, neural network research was halted in 1969 for almost 6 years when Minsky and Papert [48] pointed out two key issues related to neural network computation machines. The first issue was the inability of a single perception network model to process XOR logic. The second issue was the lack of a powerful computer to process a large neural network. In 1975 Kunihiko Fukushima [49] designed cognition, which is a precursor to the trainable multilayered neural network. Later, rediscovery of a back propagation algorithm for neural network training [50] popularized neural network research and applications in engineering problem solving. There have been many examples of applying ANN in determining COD and organic components dissolved within wastewater [37, 43-45]. In this section, a general ANN concept and two different methods of applying ANN to wastewater analysis have been discussed.

2.3.1 General ANN Conception

ANN was developed to coarsely mimic the mechanisms of the human brain. The human brain consists of billions of specialized cells called neurons. Fig. 14 shows a schematic structure of a neuron.

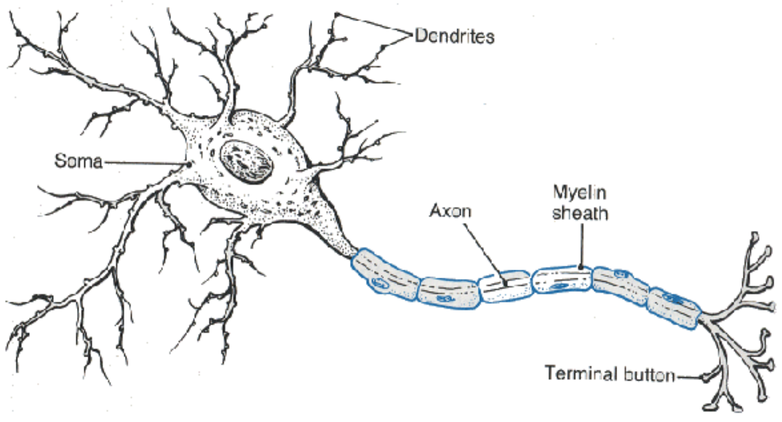


Fig. 14. Schematic Diagram for a Typical Neuron Cell

Source: Carlson, Niel. A. (1992). *Foundations of Physiological Psychology*. Needham Heights, Massachusetts: Simon & Schuster. pp. 36

A neuron has a cell body called a soma, short signal receiver branches named dendrites, and a long transmitter branch called an axon. Each cell receives electrical signals from neighboring transmitter neurons through the dendrites, and then it processes the electrical and chemical signals before sending them out through its axon via the terminal buttons and across the synaptic cleft to the neighboring receiver neurons. Through a very complex network of billions of neurons, human brain processes surrounding information received through five sensory organs: eyes, nose, ears, tongue, and skin. All the creative thinking and logic processing in a human brain is done via this intricate network of neurons.

Similar to the natural neuron, a mathematically composed artificial neuron and neural networks have been modeled and successfully used in many areas [18]. A neuron in the context of an ANN is a node *M* in a directed graph structure with a single weighted output arc and one or more multiple weighted input arcs as shown in Equation 7.

$$y = f\left(\sum_{i=0}^{n} (N_i * w_i)\right) \qquad \text{Equation 7}$$

In Equation 7 the term N_i is the output value of the *i*th input nodes for *M*, and w_i is the weight of the arc of the connection ($N_i \rightarrow M$). The function *f* can be any type of activation function, including a step, linear, or sigmoid function, depending on the application. In some cases, even stochastic functions are used to model complex behaviors in an ANN. Fig. 15 shows the typical activation function types.

An ANN, in the most ordinary sense, is a network of artificial neurons. It is a directed graph with weighted arcs, a set of input nodes, and a set of output nodes. An ANN with *N* input nodes (sometime called features) implements an approximation function with *N* independent variables. This approximation function theoretically fits well into all possible outcomes for any combination of *N* independent features.

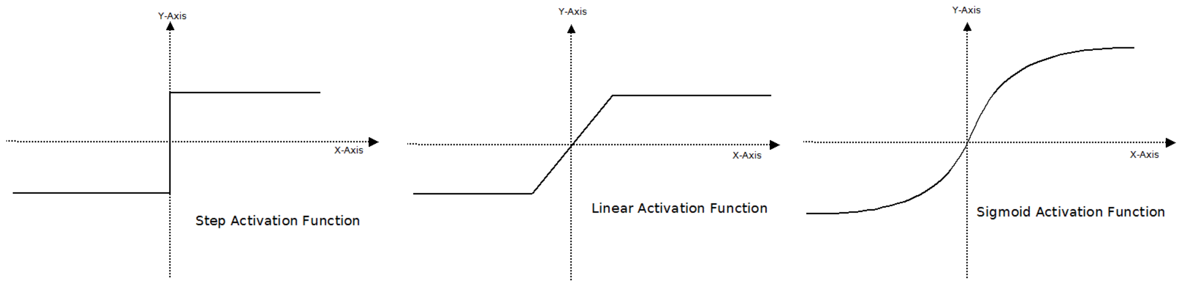


Fig. 15. Typical Activation Functions Used in ANN

Fig. 16 shows an example of an ANN graph structure and an imaginary function that an ANN is implementing. This example of an ANN has one input node (*N1*) and one output node (*N8*), which implement a function of *f*(*x*) with a single independent feature/variable *x* (which is the value of *N1*). This ANN has two hidden layers with four (*N2–5*) and two nodes (*N6–7*), respectively. Each has connected input/output arcs with different connection weights.

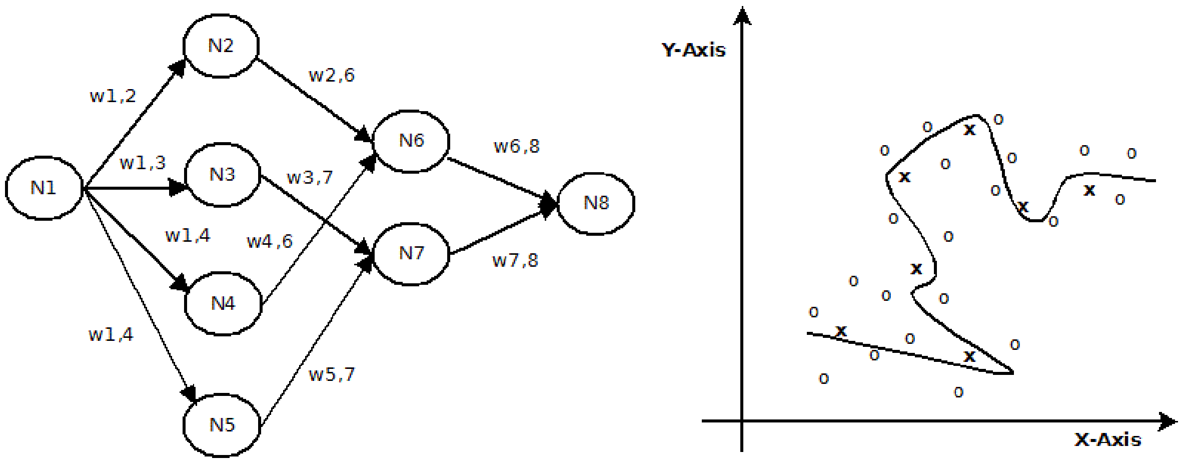


Fig. 16. Example of an ANN Graph and the Implemented Imaginary Function

In this example, the ANN graph does not contains any loops. However, the ANN structure allows loops if needed for the target application. The ANN structure without loops, as in the example, is known as a "feed-forward ANN." An example plot of a one-dimensional function implemented by this ANN is shown in Fig. 16. Possible values that the independent variable can assume and the corresponding function values are plotted as *o* and *x*. The points, marked with an *x*, are the known samples of the population. These known samples are used as a training set for the ANN. Training an ANN involves adjusting the arc weights in such a way that the error of the output values from the ANN is minimized with respect to the training set. The most common training method is the back propagation algorithm [50].

2.3.2 Application of ANN in Wastewater Analysis

Many studies show that any organic material dissolved in a water sample strongly absorbs ERM in the UV range. Different waste materials absorb a different set of wavelengths. Absorption also depends on the amount of material present. Thus, absorption analysis should have a direct relation to the COD parameter, which characterizes the wastewater contamination level. However, because many different chemicals can be part of the wastewater, analysis should not be performed on only a single or only a couple of wavelengths. The entire UV-Vis wavelength range should be analyzed with lower precision. Hence, the number of independent analysis parameters is great. This poses a challenge to conventional mathematical regression, as well as ANN training. An ANN with a large number of input nodes (one per wavelength considered) usually raises problems in training convergence. Therefore, the main challenge in ANN-based COD analysis is to reduce the number of independent parameters or features while maintaining an acceptable level of accuracy. Fogelman et al. described such a feature extraction method in the article "Neural Computing & Application" [37]. In a different application, Benjathapanun et al. showed a feature reduction technique using the second differential (or the absorption shape) of spectral absorption data [38].

CHAPTER THREE

OVERALL METHODOLOGY USED IN THE THESIS

3.1 Objectives

There were multiple objectives of this specific thesis work in relation to COD analysis on an embedded and cloud-enabled device.

1. The first objective was to establish a new methodology to identify a specific photometry pattern for water that has COD.
2. The second objective was to build a cloud-enabled embedded platform that could be used to collect photometry data from a water sample and use the developed methodology to determine if the sampled water has COD.

3.2 Method to Detect COD Patterns

Photometry data present specific patterns for water samples with COD or without COD. The basic idea for this study was to develop an algorithm to identify patterns of the photometry data for a water sample, which are deterministically indicative of COD content. Because the collected data can have random noises or variations from the data analysis environment, it is important to establish a good methodology to determine a photometry pattern with such variations and noises indicative of COD content in water. Because there can be many unknown and nondeterministic pattern-variation factors, an ANN is the preferred tool to handle such variations in collected photometry data.

A simple methodology was implemented in this thesis work. The basic idea was to divide the obtained photometry absorption data into *N* finite, equidistance, normalized samples. These *N* samples were normalized with respect to the highest data present in the set (i.e., by taking the highest data present as 1.0). An ANN was trained using these *N* samples. The implemented ANN had *N* inputs and 1 output. For this work, a single-perception ANN was used with 20 inputs and 50 intermediate nodes in the hidden layer and single output. The output state of this ANN (1 or 0) would represent the presence or absence of COD in water. Therefore, the photometry spectrum range was divided into 20 equidistant points (as shown in Fig. 17) for the sampling.

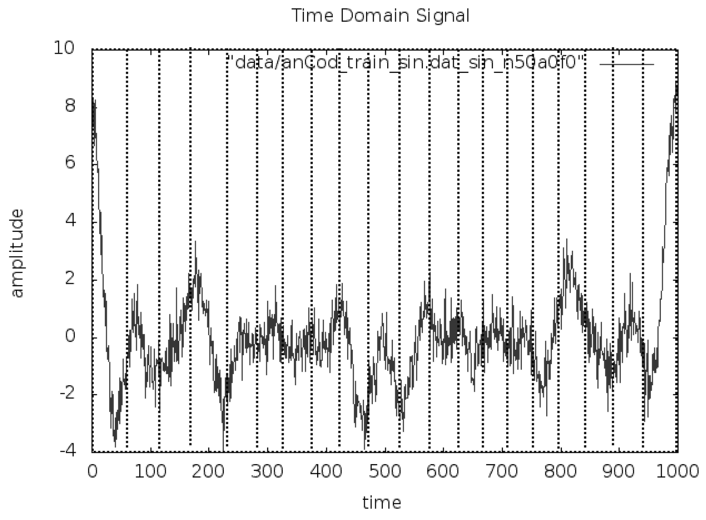


Fig. 17. Sampling the Signal

3.3 Methodology Experiment

To prove that an ANN could be deployed to identify specific photometry patterns, data were generated from a mathematical model of a complex signal and applied for pattern detection (see Equation 8) instead of applying the ANN-based pattern recognition on real photometry data.

$$F'(t) = F(t) + n(t)$$

Equation 8

Where $F(t)$ is the summation of sine harmonics with a symmetric amplitude as in Equation 9 and $n(t)$ is the uniformly distributed white noise with a probability distribution function as given in Equation 10.

$$F(t) = \sum_{n=1}^{P} (a_n \sin(2\pi\mathrm{n}\mathrm{f}\mathrm{t} + \varphi) + a_n \sin(2\pi(2\mathrm{P} - \mathrm{n} - 1)\mathrm{f}\mathrm{t}) + \varphi)$$

Equation 9

$$PDF_{n(t)} = \frac{1}{2\pi f}$$

Equation 10

The φ in Equation 9 is the random phase shift between 0 and $\pi/2$. The a_n in Equation 9 is the amplitude for the nth and the $(2\mathrm{P} - n - 1)$th harmonics of the composite signal.

The experiment was done with a P value of 1, 2, 4, 8, 16, 32, 64,128, 256, and 512, along with a base signal of $A * \sin(2\pi\mathrm{ft} + \varphi)$. The base frequency for the experiment was taken as 1 KHz. The amplitude of the symmetric harmonics was taken randomly between 0 and 1 with two extreme component amplitudes as unity. For the base signal and the signal corresponding to each P, 100 different amplitude patterns were used in the experiment. For each of the amplitude patterns, a training set was generated with 100 samples injecting a random noise and phase shift. The random noise amplitude was fixed at 20% of the base amplitude of the signal. This set was the confirmative training that should result in an ANN output of 1. Per the amplitude pattern, the training set also contained 100 random signal patterns, which were treated as non-confirmative training and should have resulted in an ANN output of 0. Because 20 equidistant sample points were taken from the white noise-injected random signal, these points also represented a nonrandom signal pattern with a different amplitude, frequency, and phase for which the ANN output should have been 0.

To generate the signal with *R* resolution, the time step was determined using Equation 11, where *f* is the frequency. Among these *R* points of the signal, covering the period *T* of the signal, *S* equidistance samples were taken to feed into an ANN input.

$$\Delta t = \frac{1}{f * R} \qquad \text{Equation 11}$$

3.4 Methodology Validation

In this experiment, *R* was set to 1,000 and *S* was set to 20. The experiment was performed using anCod software with automated scripts to generate different signal patterns. Per the amplitude pattern, 200 test sets were also generated with 100 confirmative and 100 non-confirmative tests. Therefore, per harmonic combination (1, 2, 4, 8 … 2048), there were 20,000 data patterns to train or test the implemented ANN. The error percentage comparing the test patterns' expected result was measured as the success of the methodology proposed. This experiment showed the following observations.

1. The minimum error was 0%.
2. The mean error was 2.21%.

3. The standard deviation was 5.24%.

4. The maximum error was 6.5%.

Fig. 18 shows the distribution of percentage of error over the entire experiment sample and deviation distribution.

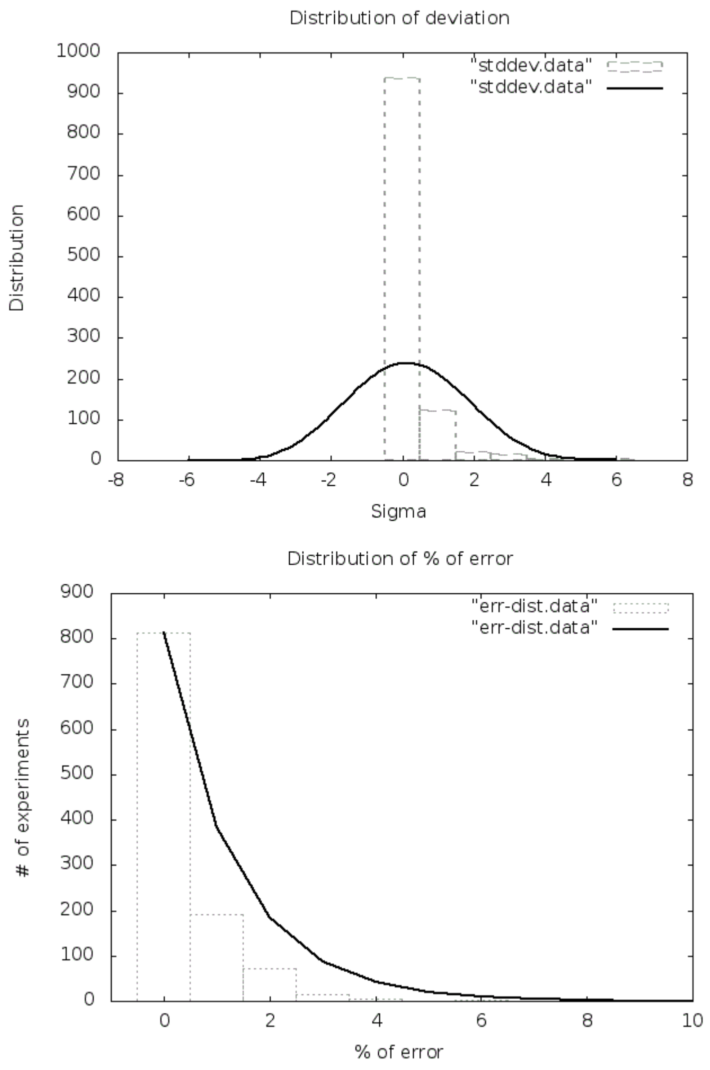
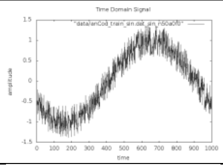
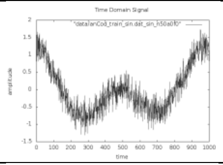
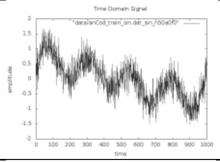
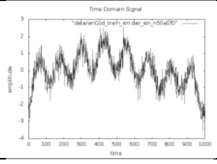
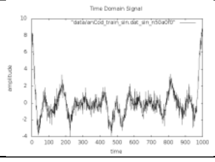
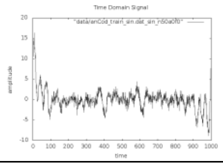
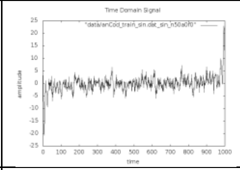
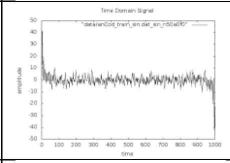
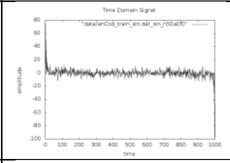
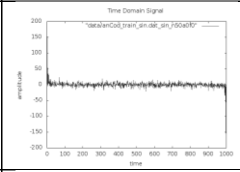
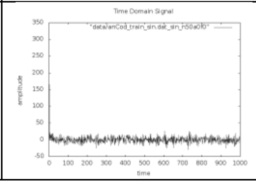


Fig. 18. Error Deviation (above) and Error Distribution (below) for the ANN

Table 2 shows the quality results of the implemented ANN and includes the number of harmonics in the composite signals and the max, min, and mean percentages of error with standard deviation. The picture of sample signal generated per harmonics is also included in Table 2.

Table 2. Methodology Experiment Results and Sample Signal

# Harmonics	Sample signal	Maximum % of error	Mean % of error	Minimum % of error	Standard deviation of the % of error
1		5.5	1.35	0.0	0.94
2		4.0	0.46	0.0	0.71
4		2.0	0.17	0.0	0.35
8		2.0	0.20	0.0	0.44
16		6.5	0.41	0.0	0.84
32		4.0	0.56	0.0	0.89

# Harmonics	Sample signal	Maximum % of error	Mean % of error	Minimum % of error	Standard deviation of the % of error
64		6.0	0.88	0.0	1.08
128		3.0	0.59	0.0	0.73
256		2.5	0.48	0.0	0.61
512		2.0	0.31	0.0	0.49
1024		2.5	0.28	0.0	0.44

3.5 Conclusion

The experiment results showed that the proposed ANN architecture and pattern recognition methodology were acceptable in terms of quality of results. The central tendency of the error was very low and within a manageable range as suggested by the mean error and standard deviation. The chart showing the distribution of error percentages (Fig. 18) also supported the same observations and conclusions.

CHAPTER FOUR

IMPLEMENTATION OF THE SYSTEM

4.1 Overall System Architecture

The target system was a cloud-based [61] mobile system armed with a portable photometer with adequate photometry analysis accuracy. Fig. 19 shows the complete system architecture in a schematic.

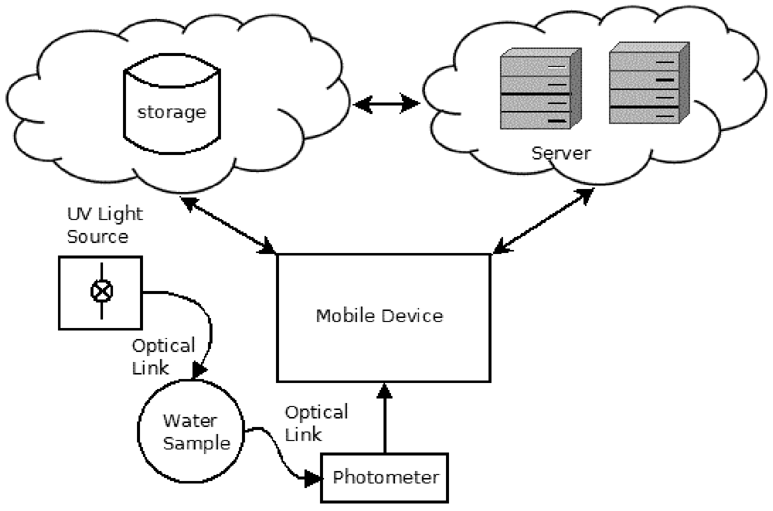


Fig. 19. Schematic of Overall System Architecture

The system contains a UV light source, water sample holder, and photometer all linked to each other through an optical fiber. The photometer is connected to the mobile device through a USB channel. The mobile device is cloud enabled to store the photometry data on the cloud and perform data analysis on the cloud server. The cloud server can also access the cloud storage to read the photometry data and store the results of the analysis. The underlying vision of this thesis was to study the possibility of constructing a cheap mobile system using a common mobile phone coupled with a portable photometer for an instantaneous COD detection of water.

4.2 Implementation Components

4.2.1 Mobile Platform

Tiny6410 [51] was selected to implement the mobile platform. This system used an S3C6410 processor, ARM11 implementation from Samsung [52], and SMDK6410 software development kit for software implementation. Fig. 20 shows the hardware board used in this work.

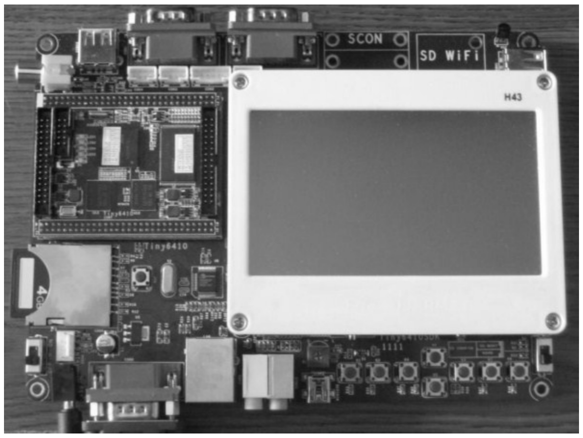


Fig. 20. Tiny6410 board

This specific system used a 4GB external SD card for system booting and storage of the mobile system. With this feature, the software footprint was of less concern than using onboard flash storage.

4.2.2 Photometer and UV Light Source

As a portable photometer, Red Tide USB650 [53] from OceanOptics was used in this work. Red Tide USB650 is a low-cost photometer for academic and college laboratory purposes. The photometer's USB connectivity provides easy connection to the Tiny6410 mobile system.

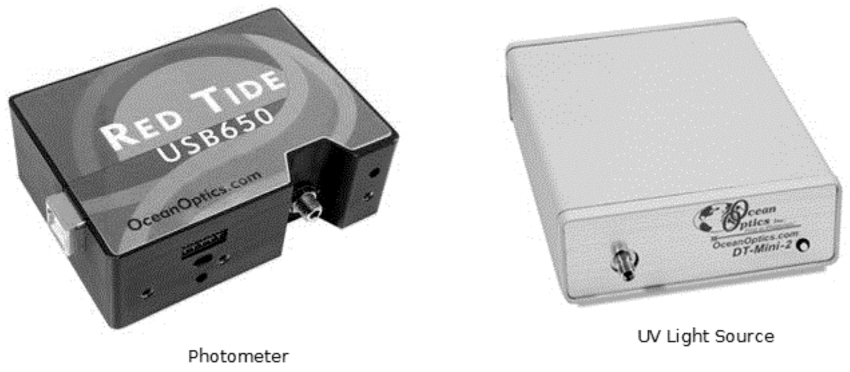
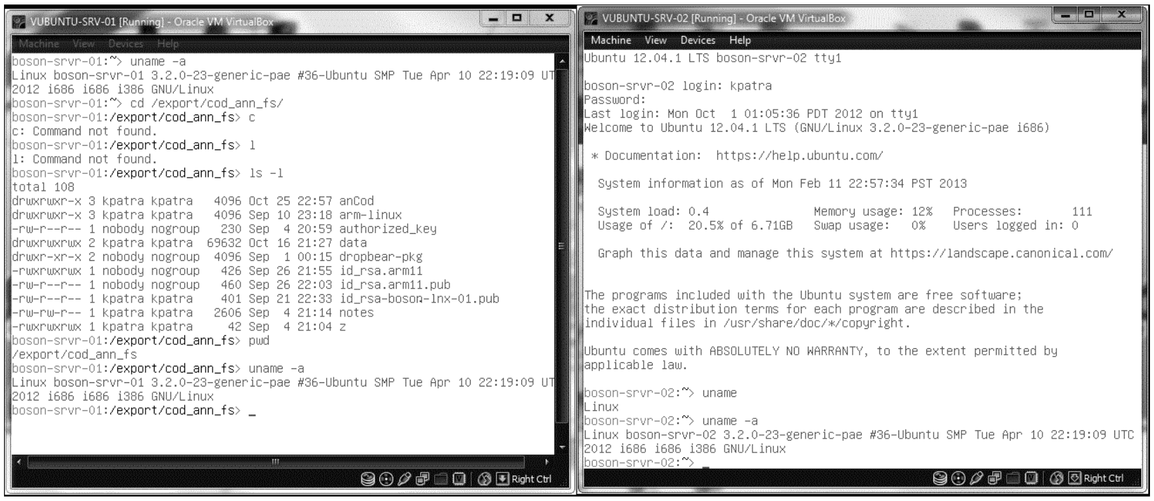


Fig. 21. Portable Photometer and UV Light Source

The DT-MINI-2 Deuterium Tungsten Halogen Light Source [72] from OceanOptics was used as the UV-Vis light source. This unit can produce light with a wavelength between 189 nm and 885 nm. Fig. 21 shows the photometer and the UV light source used in this work.

4.2.3 File and Application Server

In general, a cloud infrastructure allows virtualized machines to share common hardware. To keep the setting simple, using VirtualBox [54] two virtual servers were configured to run the UBUNTU Linux Server [55]. One of the virtual servers was configured to act as the network file server (NFS) and the other was configured as an application server receiving remote execution requests using secured shell (SSH) protocol. These servers were connected through a home router implementing a local area network (LAN).


VUBUNTU-SRV-01 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Right Ctrl

```
boson-srvr-01:~> uname -a
Linux boson-srvr-01 3.2.0-23-generic-pae #36-Ubuntu SMP Tue Apr 10 22:19:09 UT
2012 i686 i686 i386 GNU/Linux
boson-srvr-01:~> cd /export/cod_ann_fs/
boson-srvr-01:/export/cod_ann_fs> c
c: Command not found.
boson-srvr-01:/export/cod_ann_fs> l
l: Command not found.
boson-srvr-01:/export/cod_ann_fs> ls -l
total 108
drwxrwxr-x 3 kpatra kpatra   4096 Oct 25 22:57 anCod
drwxrwxr-x 3 kpatra kpatra   4096 Sep 10 23:18 arm-linux
-rw-r--r-- 1 nobody nogroup   230 Sep  4 20:59 authorized_key
drwxrwxrwx 2 kpatra kpatra  69632 Oct 16 21:27 data
drwxr-xr-x 2 nobody nogroup  4096 Sep  1 00:15 dropbear-pkg
-rwxrwxrwx 1 nobody nogroup   426 Sep 26 21:55 id_rsa.arm11
-rw-r--r-- 1 nobody nogroup   460 Sep 26 22:03 id_rsa.arm11.pub
-rw-r--r-- 1 kpatra kpatra    401 Sep 21 22:33 id_rsa-boson-lnx-01.pub
-rw-rw-r-- 1 kpatra kpatra   2606 Sep  4 21:14 notes
-rwxrwxrwx 1 kpatra kpatra     42 Sep  4 21:04 z
boson-srvr-01:/export/cod_ann_fs> pwd
/export/cod_ann_fs
boson-srvr-01:/export/cod_ann_fs> uname -a
Linux boson-srvr-01 3.2.0-23-generic-pae #36-Ubuntu SMP Tue Apr 10 22:19:09 UT
2012 i686 i686 i386 GNU/Linux
boson-srvr-01:/export/cod_ann_fs> _
```

VUBUNTU-SRV-02 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Right Ctrl

```
Ubuntu 12.04.1 LTS boson-srvr-02 tty1

boson-srvr-02 login: kpatra
Password:
Last login: Mon Oct  1 01:05:36 PDT 2012 on tty1
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-23-generic-pae i686)

 * Documentation:  https://help.ubuntu.com/

  System information as of Mon Feb 11 22:57:34 PST 2013

  System load: 0.4               Memory usage: 12%   Processes:       111
  Usage of /:  20.5% of 6.71GB   Swap usage:   0%    Users logged in: 0

  Graph this data and manage this system at https://landscape.canonical.com/


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

boson-srvr-02:~> uname
Linux
boson-srvr-02:~> uname -a
Linux boson-srvr-02 3.2.0-23-generic-pae #36-Ubuntu SMP Tue Apr 10 22:19:09 UTC
2012 i686 i686 i386 GNU/Linux
boson-srvr-02:~> _
```

Fig. 22. VirtualBox-based Virtual UBUNTU Linux Server

4.2.4 Development Environment

The development environment in this work was a mixture of Windows (as the host platform) and a virtual machine running UBUNTU Linux on a Windows host. On the Windows side, Tera Term [56] was used as the console for Tiny6410. This enabled the use of a host computer to control and operate the Tiny6410 without using Tiny6410's native small LCD display with a touch-screen keyboard. In addition, SpectraSuit [57] software from OceanOptics Inc. was used to validate the collected photometry data against the data collected from the Tiny6410 platform. This ensured that the data collection quality was within an acceptable range on the Tiny6410 platform.

A virtual machine running UBUNTU Linux was used to develop the software to control the system operation and implement the ANN-based algorithm for COD detection from the photometry data of a water sample. Netbeans [58] was used as an integrated development environment (IDE) in this work. The software was written in C and shell script to keep the memory footprint of the entire software manageable and within the limits of the memory available on a mobile platform. The GNU C native compiler for x86 was used along with a cross compiler for ARM11 (which came with the SDK). The implemented software was executable both on an x86 and an ARM11-based system through shell scripting and cross compilation.

4.3 Software Implementation Details

To keep the setup simple, an NFS-based file system was used instead of a cloud-based file system, and a secured remote shell (SSH)-based remote execution was used instead of a cloud-based remote execution. These basic changes were sufficient to prove the concept of the software, as cloud-based services are nothing but accounted remote storage and servers. With these basic changes, the top-level architecture looks similar to that shown in Fig. 23.

At first, the ARM11 system board was installed with a compiled Linux kernel and a simple GUI-based user environment. (The details of this implementation can be found in APPENDIX A.) The photometer came with an application and driver for the Windows platform. In addition, a device driver module was developed and installed on the Linux side to gather photometry data on the ARM11 platform. Previous internal work [59] performed at the San Jośe State University Computer Engineering Department was used for this purpose. (Details can be found in APPENDIX B.)

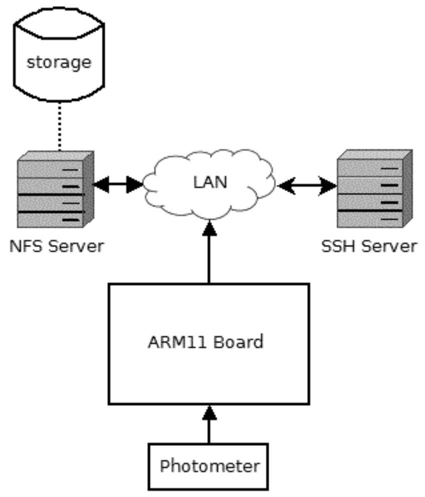


Fig. 23. Simplified System Architecture Replacing Cloud

The virtual servers for NFS and SSH-based remote application executions were configured using the UBUNTU Linux server installed on VirtualBox and running on the host Windows system. Both of the servers were connected to each other using a LAN implemented using a domestic router. (Details of this implementation can be found in APPENDIX C.)

To implement the ANN, a fast artificial neural network (FANN) library [60] was used. (An introduction to this library can be found in APPENDIX D.) The existing library can store trained ANN models in a text file that may be loaded into the execution memory using the FANN library API. However, the memory footprint for this library is quite large; thus, it is not suitable for use on a mobile platform. To use the FANN library effectively to train ANN for this thesis work, the library was extended to generate a compact, precompiled C model for the trained ANN. In that way, the ANN model can be loaded into the memory and used in the target application without loading the entire FANN library. (The details of this FANN extension for a precompiled C model generation can be found in APPENDIX E). Finally, the anCod software system was implemented to control the system operation. (The details of this software can be found in APPENDIX F). Fig. 24 depicts the complete ecosystem implemented in this work.

In the implemented system, the ARM11 board hosted the anCod client process, which controlled the operation of the UV light source and the photometer's data-capturing operation. This client also processed the photometry data and stored the processed data on an NFS partition. The captured data were either training data with a known COD for ANN training purposes or test data with an unknown COD. These two types of data were created depending on the options used in an anCod client invocation for a test or training data set generation. This client process also requested remote execution to the SSH server to run the anCod trainer or server, depending on the option provided at run time.

The anCod trainer process was executed remotely on the SSH server. It read the training data stored in the NFS partition and created a compiled ANN model after training was completed. The anCod server process was also executed on the remote SSH server. It analyzed the test photometry data stored on the NFS partition by the client process using the trained ANN stored by the trainer process. The same NFS partition stored the analysis results, and the client process used this stored result to display the results through the ARM11 system.

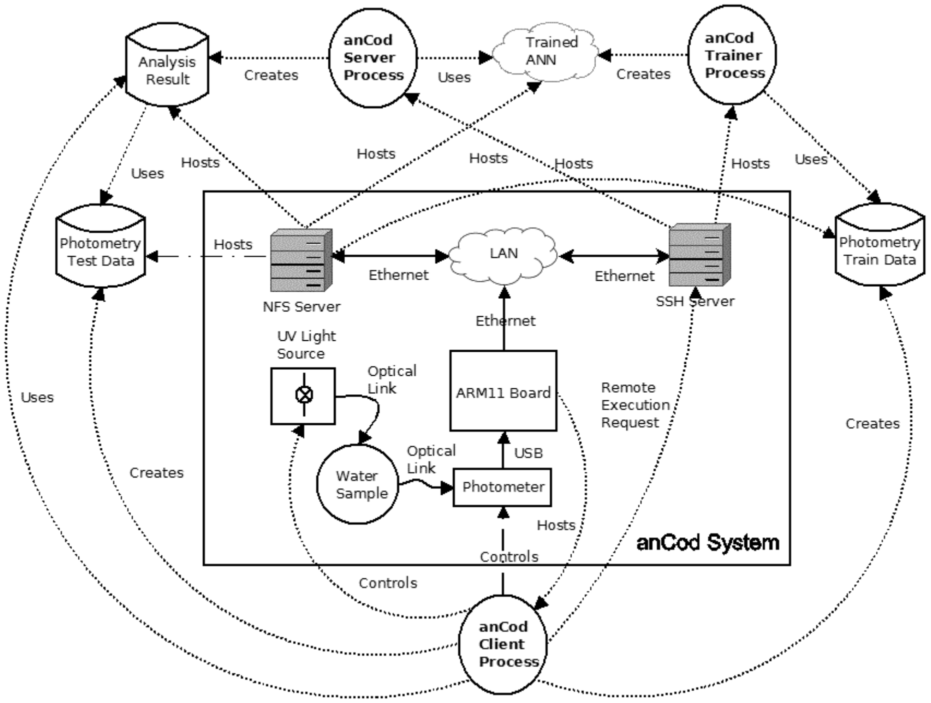


Fig. 24. anCod Hardware and Software Ecosystem

4.4 System Implementation

Fig. 25 shows the actual setup for the overall system. There were two distinct parts of the system. One was the computing system, which analyzed the photometry data. The other part was the chemistry and photonics laboratory setup. This experiment used very basic chemistry laboratory sets. The setup consisted of a 1L laboratory flask (to store the calibrated sample for COD measurement), a 100 ml graduated cylinder, test tubes (to hold various concentrations of COD samples), test tube holders, funnels, and tubes. The photonics part of the laboratory setup consisted of a UV-Vis light source, fiber channel connectors, and the flow cell.

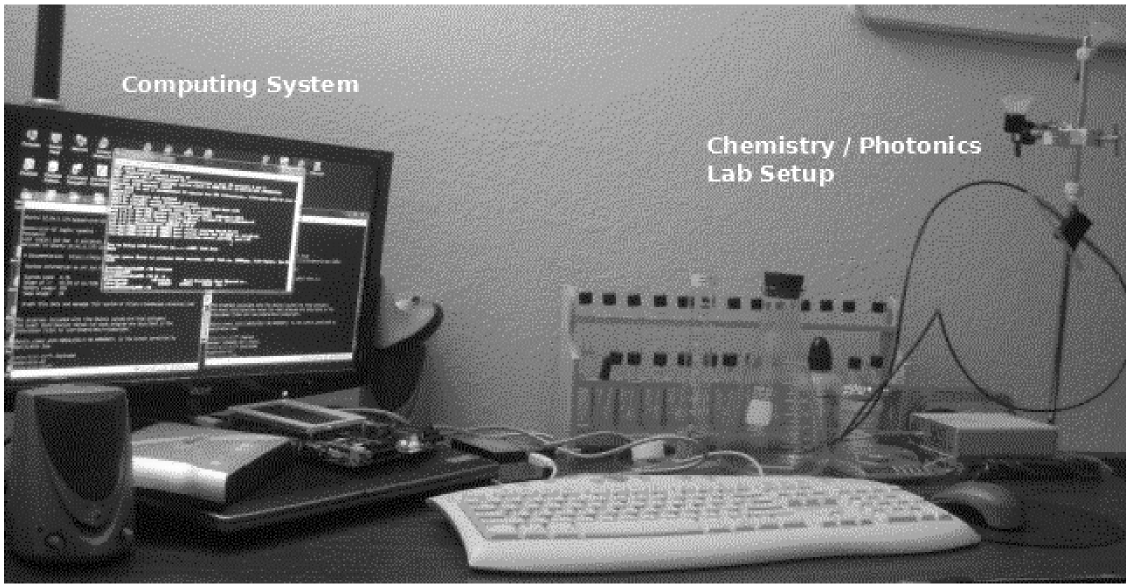


Fig. 25. Overall Setup for the COD Analysis System

The photonic lab setup used gravity and the water's property of being at the same level to conduct the experiment. Fig. 26 shows the setup for the photonics lab part in this experiment along with the flow cell picture. The entire sample flow system was built on a chemistry lab stand. The water sample was injected into the flow cell through the receiver funnel, which was connected to the flow cell's sample inlet though a tube. The flow cell was kept in such a position that water could always flow through the capillary channel inside the flow cell under the influence of gravity. During the sample analysis, the draining funnel was kept high using a test tube holder, which was also connected to the flow cell's sample outlet through another tube.

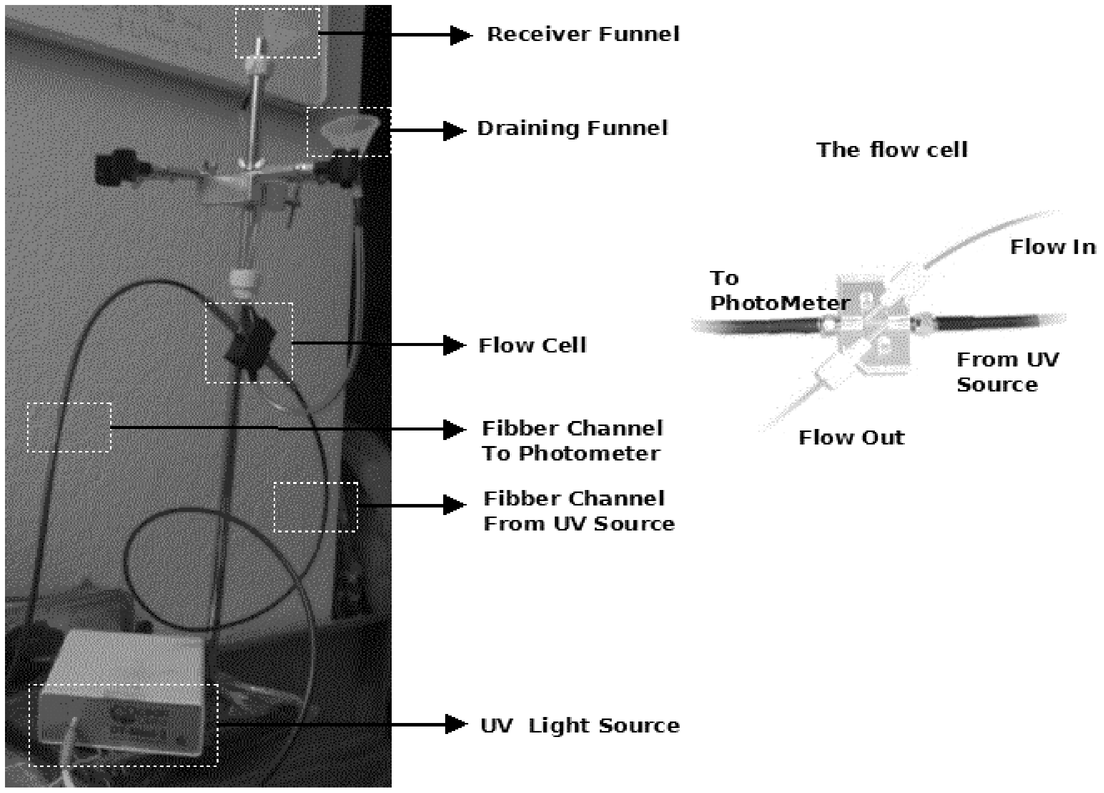


Fig. 26. Photonics Lab Setup and Flow Cell

[Source: The flow cell photo was taken from the OceanOptics website http://www.oceanoptics.com/Products/fiazsmaflowcells.asp]

The flow cell was wrapped with black tape to reduce interference from surrounding light sources. The fiber channel cable from the UV light source was connected to one end of the flow cell and another fiber cable to the photometer was connected to the other end of the flow cell. When the sample analysis was done, the draining funnel could be lowered and then the sample could be drained by gravity. Because the flow cell channel was a capillary, an external hand air pump was used to make sure the capillary channel drained properly.

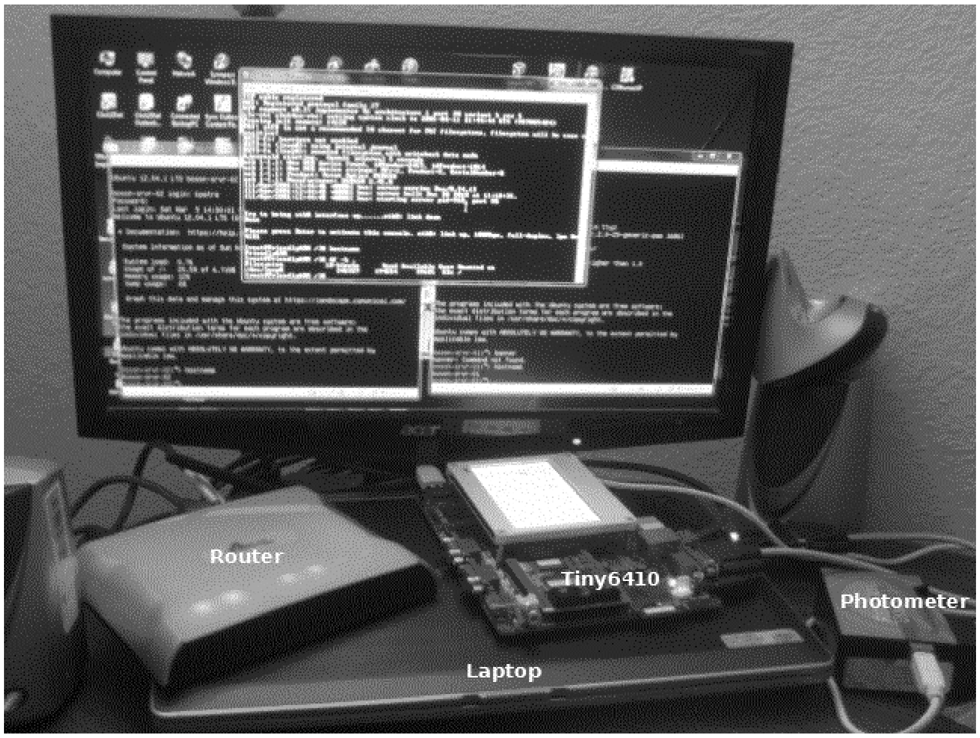
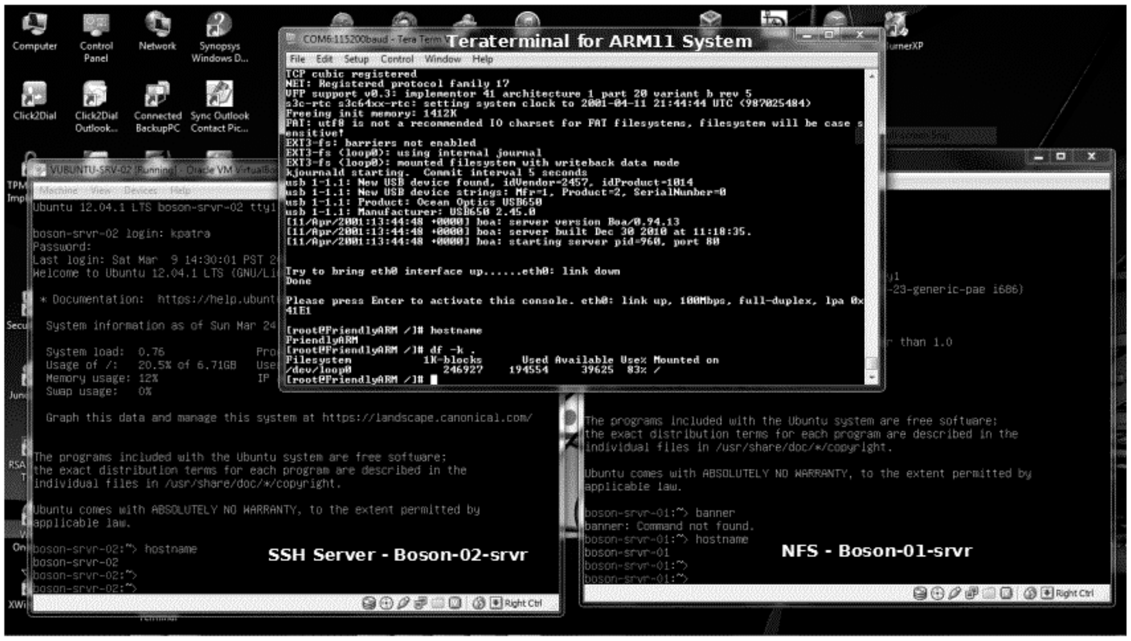


Fig. 27. Computing Setup for COD Analysis

Fig. 27 shows the computing setup for COD analysis. The entire thesis work used a laptop (Dell Latitude with 4x2.7 GHz x64 CPU, 4GB RAM, and a 120 GB hard disk) as a computer platform. The mobile platform Tiny6410, connected with the photometer (Red Tide USB650) through a USB, was connected to the laptop on a LAN through a home-based router.

The laptop hosted two virtual servers running under VirtualBox and the HyperTerminal for the Tiny6410. Fig. 28 shows the three running windows for the system connected through the LAN. One of the virtual servers served as an NFS that hosted a disk partition and was accessible to the Tiny6410 and any other machine on the LAN.


Teraterminal for ARM11 System
COM6:115200baud - Tera Term
File Edit Setup Control Window Help
SSH Server - Boson-02-srvr
NFS - Boson-01-srvr
VUBUNTU-SRV-02 [Running] - Oracle VM VirtualBox
Machine View Devices Help
Right Ctrl

```
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev 5
s3c-rtc s3c64xx-rtc: setting system clock to 2001-04-11 21:44:44 UTC (987025484)
Freeing init memory: 1412K
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case s
ensitive!
EXT3-fs: barriers not enabled
EXT3-fs (loop0): using internal journal
EXT3-fs (loop0): mounted filesystem with writeback data mode
kjournald starting.  Commit interval 5 seconds
usb 1-1.1: New USB device found, idVendor=2457, idProduct=1014
usb 1-1.1: New USB device strings: Mfr=1, Product=2, SerialNumber=0
usb 1-1.1: Product: Ocean Optics USB650
usb 1-1.1: Manufacturer: USB650 2.45.0
[11/Apr/2001:13:44:48 +0000] boa: server version Boa/0.94.13
[11/Apr/2001:13:44:48 +0000] boa: server built Dec 30 2010 at 11:18:35.
[11/Apr/2001:13:44:48 +0000] boa: starting server pid=960, port 80


Try to bring eth0 interface up......eth0: link down
Done

Please press Enter to activate this console. eth0: link up, 100Mbps, full-duplex, lpa 0x
41E1

[root@FriendlyARM /]# hostname
FriendlyARM
[root@FriendlyARM /]# df -k .
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/loop0              246927    194554     39625  83% /
[root@FriendlyARM /]#
```

```
Ubuntu 12.04.1 LTS boson-srvr-02 tty1

boson-srvr-02 login: kpatra
Password:
Last login: Sat Mar  9 14:30:01 PST 2
Welcome to Ubuntu 12.04.1 LTS (GNU/Li

 * Documentation:  https://help.ubunt

  System information as of Sun Mar 24

  System load:  0.76              Pro
  Usage of /:   20.5% of 6.71GB   Use
  Memory usage: 12%               IP
  Swap usage:   0%

  Graph this data and manage this system at https://landscape.canonical.com/


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

boson-srvr-02:~> hostname
boson-srvr-02
boson-srvr-02:~>
boson-srvr-02:~>
```

```
y1
-23-generic-pae i686)

r than 1.0

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

boson-srvr-01:~> banner
banner: Command not found.
boson-srvr-01:~> hostname
boson-srvr-01
boson-srvr-01:~>
boson-srvr-01:~>
```

Fig. 28. Screen Capture of the Running Instances

The other virtual server acted as a remote execution platform for the Tiny6410. This server used SSH protocol for remote execution. The Tiny6410 system could access this machine to run any analysis remotely and obtain the analysis results.

CHAPTER FIVE

EXPERIMENT, RESULTS, AND CONCLUSION

5.1 Experiment Setup

The system operation and validity were established through several steps by performing COD detection in known samples. A set of calibrated water samples with a known COD amount was prepared. The calibrated sample preparation method is described in subsection 5.1.1. The spectroscopy data were collected for these calibrated samples and used in ANN training. Random noise was injected into the collected data to produce realistic data in a natural environment. The trained ANN produced an affirmative result of 1 for spectroscopy data collected from a water sample with COD and produced a result of 0 otherwise. The trained ANN was then tested with the collected data and the sample of water from different sources. The success of these tests should prove the validity of the overall system. The following subsections discuss each of the experiment steps in detail.

5.1.1 Calibrated Sample Preparation for COD Measurement

"The Industry Standard COD testing is based upon the theoretical amount of oxygen required to oxidize organic compounds to CO_2 and H_2O" [62]. This means that to establish the efficiency of any COD measurement instrument or methodology, a standard solution of known COD is prepared and tested using the subject of interest (e.g., an instrument or methodology). Potassium hydrogen phthalate (KHP) is the most commonly used compound to prepare samples of known COD [62].

The theoretical oxygen demand can be shown as the chemical balance in Equation 12.

$$KC_8H_5O_4 + 7.5\ O_2 \ \rightarrow 8\ CO_2 + 2H_2O + KOH \qquad \text{Equation 12}$$

Equation 12 demonstrates how each molecule of KHP is completely digested with 7.5 molecules of oxygen. In terms of weight, 1.175 mg of oxygen is required to digest 1 mg of KHP. Hence, to prepare a solution with *x* mg COD per liter of water, *x* / 1.1.75 mg of KHP must be dissolved per 1 liter of water. For example, to prepare a 1000 mg COD per liter of water, approximately 1702 mg of KHP per 1 liter of water is needed (see Equation 13).

$$\frac{1000\ mg\ per\ liter\ COD}{1.175\ mg\ O_2\ per\ mg\ KHP} = 851.1\ mg\ KHP\ per\ liter \qquad \text{Equation 13}$$

A calibrated sample of 1 L water solution with 1000 mg/L COD was prepared for the experiment. The preparation method followed Hach's booklet [62]. A precise amount of 851.5 mg of KHP was first dried overnight at 120^0C. This dried KHP was then mixed with 1000 ml organic-free deionized water using Class A glassware.

5.1.2 UV Spectroscopy Data Collection

The spectroscopy data were collected using the anCod software in training mode. Data were collected using a spectrometer connected to the embedded system. For the experiment, samples with different CODs were used. A set of samples with COD were used for ANN training (0, 250, 500, and 1000 mg/L). To prepare the training samples, the base solution of 1000 mg/L COD was diluted with the required amount of organic-free deionized water to prepare a 100 ml solution with targeted COD. Table 3 shows the amount of calibrated solution mixed with diluted water. A diluted solution was prepared by inverting the mixture 10 times.

For testing purposes, the following three samples of water were collected and then spectroscopy data were collected.

1. Filtered drinking water
2. Toilet flush system water
3. A mixture of a very small amount of the calibrated solution into 10 ml of drinking water.

Table 3. COD Solution Preparation Amounts

COD (mg) / L	Amount of calibrated solution (ml)	Amount of diluted water (ml)
0	0	100
250	25	75
500	50	50
1000	100	0

Spectroscopy data for each sample were taken five times for ANN training and testing in order to take into account the natural variation of data collected under variable conditions. Hence, 35 photometry data sets were collected. All the spectrometry data were normalized with respect to their highest values. Therefore, the data values fed into ANN training and testing within a range of 0–1.

A small script was written to call anCod with appropriate run-time arguments for data collection. The script used was named ancod_data_collection.csh.

```
#!/bin/tcsh
setenv ANCOD_USER_CONFIG_FILE ./script/ancod_data_collection_config.sh
anCod -save_as pm_${1}_${2}.dat
```

The external configuration file used was named ancod_data_collection_config.csh, which had the following content.

```
export anCod_data_dir="./data"
```

This file overrode the default directory of data storage. The script called anCod with the argument “save_as,” which prompted the photometer to collect data and store them into the data/pm_<arg1>_<arg2>.dat file. For each COD sample, the script was invoked as the following commands in the collected data file named pm_<cod>_<index>.dat.

```
$> ./script/ancod_data_collection.csh 0 1 # COD=0, Index=1
$> ./script/ancod_data_collection.csh 1000 5 # COD=1000, Index=5
```

5.1.3 ANN Training with Collected Data

Because the basic target of this work was to develop a system to detect COD in water and not to measure the amount of COD, the spectroscopy pattern from a sample with 0 COD (or the pure organic-free deionized water) was used for ANN training to produce a negative indication of 0. The outcome of 0 from the ANN evaluation represented an absence of COD. Hence, any spectrometry data that produced 0 from the ANN could be declared organic free water. The spectroscopy data from a water sample with COD were fed for ANN training to produce a result of 1. Hence, any spectrometry data producing 1 from ANN could be declared water with an organic pollutant.

For training purposes, spectrometry data were collected for water samples with 0, 250, 500, or 1000 mg/L COD. Each water sample had five collected photometry patterns, which meant that there were 20 photometry patterns. Each of these 20 patterns was used as the base pattern to generate nine more patterns with 10% random noise. This generated 200 patterns to be used in ANN training.

A script was written to train ANN with the collected photometry data. This script first generated ANN training patterns and then called the anCod trainer to create and train ANN. The following was the source code for the script executed to train ANN:

```
# Script: ann_train.csh

#!/bin/tcsh
setenv ANCOD_USER_CONFIG_FILE ./script/ann_train_config_no_random.sh
```

```
set log="log.train"


/bin/rm -f ${log}

/usr/bin/touch ${log}

/bin/rm -f ./data/anCod_ann_train.dat


# Set the parameters

set cod=0

set index=1

set pm_data_dir="../data_collection/data"




# Generate the training patterns that would give 0.0 as result

while ($index < 6)

    anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 0.0 -local
|& tee -a ${log}

    @ index = $index + 1

end


# Generate the training patterns that would give 1.0 as result

foreach cod (250 500 1000)

    set index=1

    while ($index < 6)

        anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 1.0 -
local |& tee -a ${log}

        @ index = $index + 1

    end

end
```

```
# Generate training pattern with 10% random noise injected

setenv ANCOD_USER_CONFIG_FILE ./script/ann_train_config_random_noise.sh



# Generate the training patterns that would give 1.0 as result

set cod=0

set i=1

set index=1


while ($i < 10)

    echo "INFO: Iteration $i of random noise injection for result 1.0
..."

    while ($index < 6)

        anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 0.0 -
local |& tee -a ${log}

        @ index = $index + 1

    end

    @ i = $i + 1

end


# Generate the training patterns that would give 0.0 as result

set i=1

set index=1


while ($i < 10)

    echo "INFO: Iteration $i of random noise injection for result 0.0
..."

    foreach cod (250 500 1000)

        set index=1
        while ($index < 6)

            anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 1.0
-local |& tee -a ${log}

            @ index = $index + 1

        end

    end

    @ i = $i + 1

end


# Train and generate the ANN

anCodTrainer -train pm |& tee -a ${log}
```

The configuration files were used with the following source code. There were three configuration files to define the configuration and generate ANN training data without any random noise injection. Another ANN training data set had 10% random noise injected.

```
# Configuration: ann_training_config_no_random.sh

.  ./script/ann_train_config.sh

# data usage parameters

export anCod_add_noise=0.0 # 0% additional noise added to data


# Configuration: ann_training_config_random_noise.sh

.  ./script/ann_train_config.sh

# data usage parameters

export anCod_add_noise=0.1 # 10% additional noise added to data
```

```
# Configuration: ann_train_config.sh
export anCod_data_dir="./data"

# ANN architecture
export anCod_ann_input=20
export anCod_ann_output=1
export anCod_ann_hidden_node=50

# ANN library information
export anCod_model_name=ann_ancod
export anCod_model_version=1.0.0
export anCod_build_tool=/usr/bin/make
export anCod_model_loc=./ann
```

ANN training in this experiment required 21 iterations to converge and complete the training using the back propagation algorithm. The following is the output snippet from ANN training.

```
.  .  .
INFO [anCodTrainer_multilayer_feedforward_ann.c@71)] : Training the ANN
...

Max epochs    10000.  Desired error: 0.0010000000.
Epochs            1.  Current error: 0.0602034740.  Bit fail 12.
Epochs           10.  Current error: 0.0037730325.  Bit fail 1.
Epochs           20.  Current error: 0.0010124878.  Bit fail 0.
Epochs           21.  Current error: 0.0009350366.  Bit fail 0.
```

```
INFO [anCodTrainer_multilayer_feedforward_ann.c@76)] : Training the ANN
...  DONE

.   .   .
```

5.1.4 ANN Testing with Test Samples

For testing purposes, spectrometry data for water samples with 0, 250, 500, or 1,000 mg/L COD are used. Along with them, water samples from drinking water, random mixtures, and toilet flush system water were also used. Each water sample had five collected photometry patterns, which means there were 35 photometry patterns. Each of these 35 patterns was used as the base pattern to generate nine more patterns with 10% random noise. This created 350 patterns to be used in ANN testing.

The following script generated testing patterns and called the program to give the test outcome.

```
# Script ann_test.csh

#!/bin/tcsh
setenv ANCOD_USER_CONFIG_FILE ./script/ann_test_config_no_random.sh

set log="log.test"
/bin/rm -f ${log}
/usr/bin/touch ${log}
/bin/rm -f ./data/anCod_ann_test.dat
/bin/cp ./data/anCod_ann_train.dat ./data/anCod_ann_test.dat
```

```
# Set the parameters

set cod=0

set index=1

set pm_data_dir="../data_collection/data"


# Generate the training patterns that would give 1.0 as result

foreach cod (250 500 1000 sw tlw)

    set index=1

    while ($index < 6)

        anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 1.0 -
local |& tee -a ${log}

        @ index = $index + 1

    end

end


# Generate the training patterns that would give 0.0 as result

foreach cod (fw kw)

    set index=1

    while ($index < 6)

        anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 0.0 -
local |& tee -a ${log}

        @ index = $index + 1

    end

end


# Generate training pattern with 10% random noise injected

setenv ANCOD_USER_CONFIG_FILE ./script/ann_train_config_random_noise.sh


# Generate the training patterns that would give 0.0 as result
```

```
set i=1


while ($i < 10)

    echo "INFO: Iteration $i of random noise injection for result 1.0
..."

    foreach cod (250 500 1000 sw tlw)

        set index=1

        while ($index < 6)

            anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 1.0
-local |& tee -a ${log}

            @ index = $index + 1

        end

    end

    @ i = $i + 1

end


set i=1


while ($i < 10)

    echo "INFO: Iteration $i of random noise injection for result 1.0
..."

    foreach cod (fw kw)

        set index=1

        while ($index < 6)

            anCod -sim ${pm_data_dir}/pm_${cod}_${index}.dat -train 0.0
-local |& tee -a ${log}

            @ index = $index + 1

        end

    end

    @ i = $i + 1
```

```
end


# Test the generated ANN
anCodServer -test pm |& tee -a ${log}
```

Each test set with 350 patterns was generated 100 times with 10% random noise injection and tested. This provided meaningful error distribution information. A script was written to automate this testing process.

```
# Script ann_test_100.csh
#!/bin/tcsh
set index=1

while ($index < 101)
   rm -f ./data/anCod_ann_test.dat
   rm -f ./data/anCod_ann_train.dat
   ./script/ann_test.csh
   mv log.test log/log_test.${index}
   @ index = $index + 1
end
```

The configuration files were used as the following source code. There were two configuration files defined. One generated ANN test data without any random noise injection. The other generated ANN test data with 10% random noise injected. These two configurations reused the training configuration but changed the training file name.

```
# Configuration: ann_test_config_no_random.sh

.  ./script/ann_train_config_no_random.sh
export anCod_train_file="anCod_ann_test.dat"



# Configuration: ann_test_config_random_noise.sh

.  ./script/ann_train_config_random_noise.sh
export anCod_train_file="anCod_ann_test.dat"
```

5.2 Results

The observed parameter for all 100 tests with random noise injection was the percentage of error where ANN failed to correctly detect if the water sample had COD or not. All these percentage errors were collected and statistically analyzed to determine the mean and standard deviation of the percentage of error. Table 4 shows the experimental results.

Table 4. Experimental Result for Methodology Validation

Maximum percentage of error	0.0
Mean of the percentage of error	0.0
Minimum of the percentage of error	0.0
Standard deviation of the percentage of error	0.0

5.3 Conclusion

Three major observations and conclusions can be made from this work:

1. COD detection is possible without using any chemicals based on pure photometry data analysis using an ANN. Fig. 29 and Fig. 30 show the photometry data patterns for samples used for training and samples used for testing. Given the sample data, the system was able to correctly predict if the sample had COD or not.

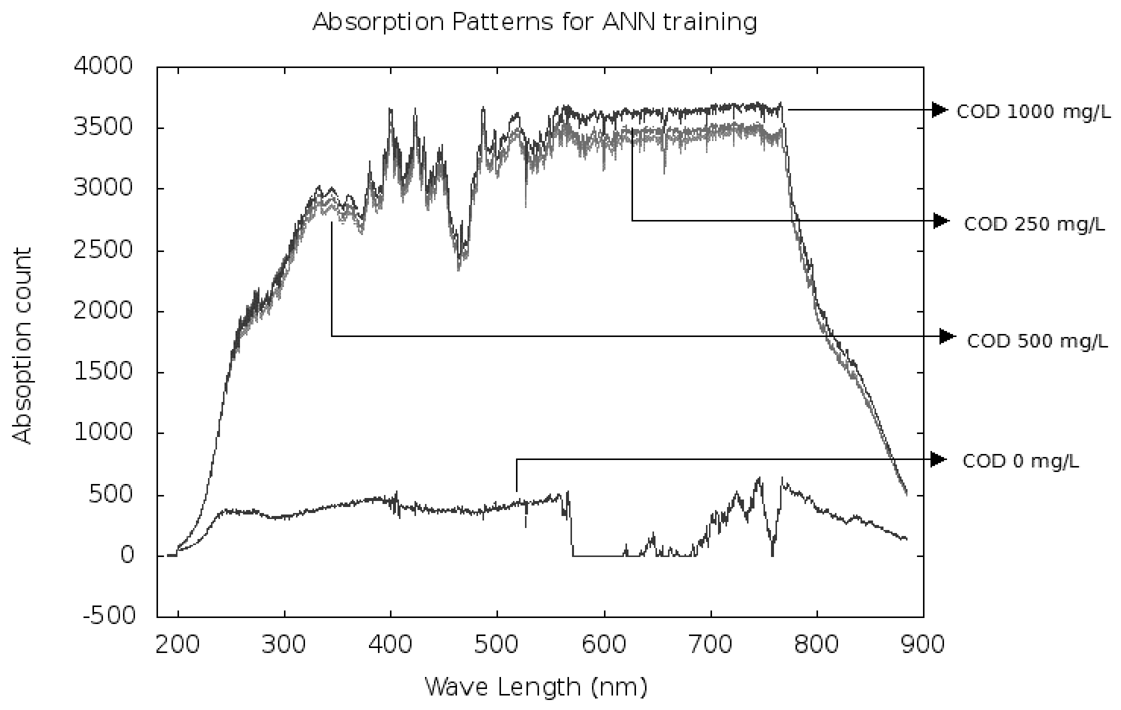


Fig. 29. Absorption Patterns for ANN Training

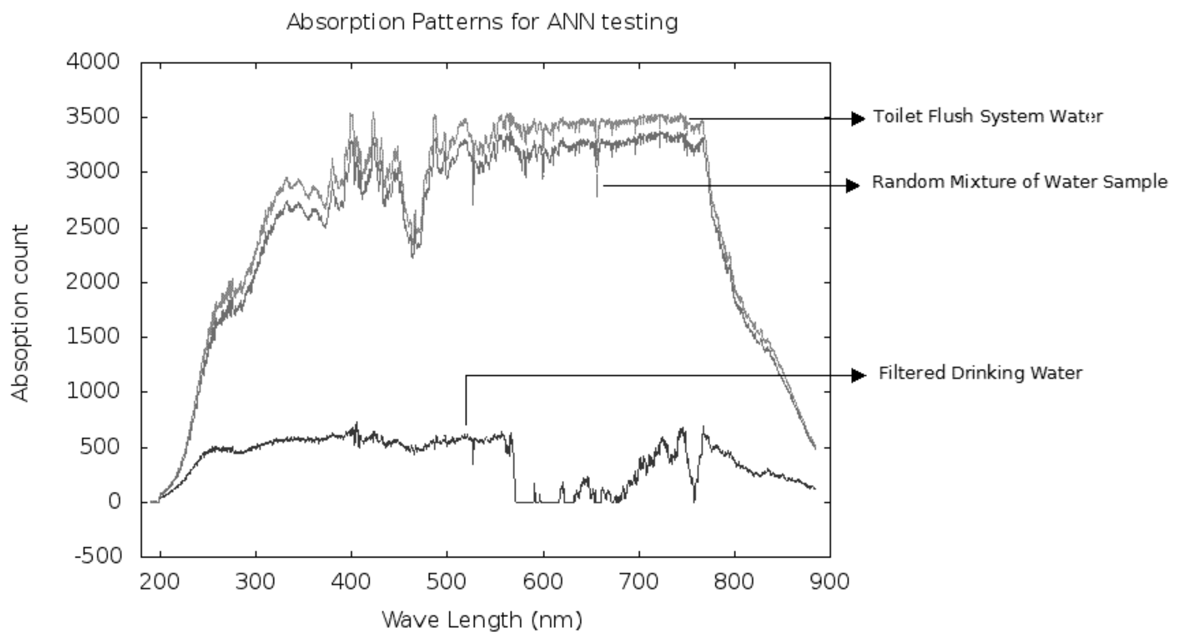


Fig. 30. Absorption Patterns for ANN Testing

2. An affordable COD detection system can be built using generic mobile embedded systems extended with an attachable spectrometer through USB or another standard connection. In this work, a spectrometer connection through a USB was used. However, this concept can be extended in order to build a Bluetooth- or Wi-Fi-enabled spectrometer to be attached to a mobile system. Henceforth, it is possible to use and write an application for a USB/Bluetooth/Wi-Fi-enabled cell phone for COD detection using a simple lightweight photometer.
3. A COD detection system can be built on top of a cloud-based infrastructure to further decrease the system cost by sharing software and hardware resources

using the cloud technique, though in this work a real cloud system was not used. However, a LAN-based infrastructure was used for remote storage and analysis of data. This same concept could be extended to a cloud-based system, which is more widely shared and accounted for through strict-usage bookkeeping.

CHAPTER SIX

FUTURE DIRECTIONS

Four future works based on this work are suggested:

1. Develop an ANN-based algorithm to analyze and compute actual COD numbers for the water sample. For this purpose, samples must be prepared to cover the entire pattern space from 0 mg/L COD to 1000 mg/L COD. Each sample can then be analyzed to identify specific absorption patterns and assigned a COD value.
2. Implement a stable experiment platform for COD measurement using photometry. A simple motor-based sample injection system and flushing system controlled by the embedded system can be designed for this purpose.
3. Integrate a real cloud infrastructure for the system. In the current system, LAN-based NFS for data storage and remote execution using SSH were used. These could be extended to use real cloud infrastructure or a private cloud to prove the concept. Either UBUNTU cloud infrastructure [63] or OpenStack [64] could be used to build a private cloud infrastructure. Specific steps to set up OpenStack can be found on the Internet [65, 66]. The client application for the cloud should be ported into the ARM11 platform for this purpose.
4. Develop a self-learning, continuously adoptable system that will dynamically train ANN with a new set of training data as available after chemical-based COD measurement. Because anCod software uses an ANN model dynamically from the installation location of the software, it is possible to replace the current ANN

with a better-trained ANN at any point of time. Hence, the system can compute the initial COD values in situ. In addition, when verified COD values are generated through a chemical process, the ANN can be retrained with extended training data. Fig. 31 explains the basic concept. In this system, the mobile anCod analysis system is used for fast analysis for the COD determination of a water sample. The same sample is also analyzed in the lab using a conventional process and the result is fed back into the anCod training system to update ANN. This way, the ANN will be retrained to handle the varieties of water samples found in real life.

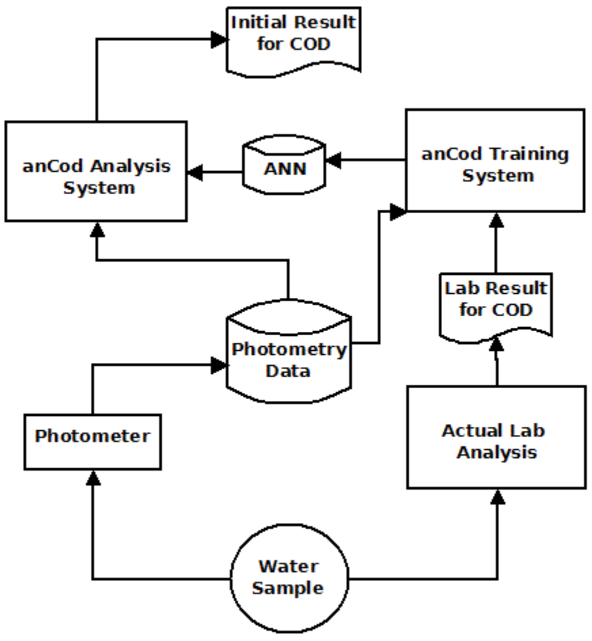


Fig. 31. Evolving anCod Analysis System with Feedback from Lab Result

References

1. Wikipedia Contributors. (2012). *Chemical oxygen demand* [Online]. Available: http://en.wikipedia.org/wiki/Chemical_Oxygen_Demand (Accessed May 18, 2012).

2. Wikipedia Contributors. (2012). *Biological oxygen demand* [Online]. Available: http://en.wikipedia.org/wiki/Biochemical_oxygen_demand (Accessed May 18, 2012).

3. U.S. Geological Survey. (2012). *How much water is there in, on and above the Earth?* [Online]. Available: http://ga.water.usgs.gov/edu/earthhowmuch.html (Accessed May 20, 2012).

4. U.S. Geological Survey. (2012). *The water cycle* [Online]. Available: http://ga.water.usgs.gov/edu/watercycle.html (Accessed May 20, 2012).

5. WHO/UNICEF, "Progress in drinking-water and sanitation: Special focus on sanitation," WHO/UNICEF Joint Monitoring Programme for Water Supply and Sanitation, July 17, 2008, pp. 25.

6. UNICEF. (2008). *JMP 2008* [Online]. Available: http://www.unicef.org/media/media_44093.html (Accessed May 21, 2012).

7. Kally Worm. *Water is life—Groundwater drawdown* [Online]. Available: http://academic.evergreen.edu/g/grossmaz/WORMKA/ (Accessed May 21, 2012).

8. Wikipedia Contributors. (2012). *Water crisis* [Online]. Available: http://en.wikipedia.org/wiki/Water_crisis#cite_ref-12 (Accessed May 21, 2012).

9. Wikipedia Contributors. (2012). *Water treatment* [Online]. Available: http://en.wikipedia.org/wiki/Water_treatment_plant (Accessed May 22, 2012).

10. Explore More Project Contributors. (2012). *Industrial uses* [Online]. Available: http://www.iptv.org/exploremore/water/uses/use_industry.cfm (Accessed May 23, 2012).

11. United States Environment Protection Agency. (2012). *Drinking water contaminants* [Online]. Available: http://water.epa.gov/drink/contaminants/index.cfm#1 (Accessed May 22, 2012).

12. B. Kiepper. (2010, Oct. 14). *Understanding laboratory waste water test: I. ORGANICS (BOD, COD, TOC, O&G)* [Online]. Available: http://www.caes.uga.edu/publications/pubDetail.cfm?pk_id=7895 (Accessed May 23, 2012).

13. H. Long. (2012). *Chemical oxygen demand (COD) testing procedures* [Online]. Available: http://water.me.vccs.edu/courses/ENV149/cod_print.htm (Accessed May 23, 2012).

14. Missouri Department of Natural Resources. (2012). *Water quality parameters* [Online]. Available: http://www.dnr.mo.gov/env/esp/waterquality-parameters.htm (Accessed May 23, 2012).

15. ManSci Inc. (2012). *COD analysis PeCOD^{TM} from MANTECH* [Online]. Available: http://www.mansci.com/products/PeCOD/index.html#WhatisPeCOD (Accessed May 23, 2012).

16. C. Zhang, *Fundamentals of Environmental Sampling and Analysis*. Hoboken, NJ: Wiley, 2007, pp. 357.

17. Wikipedia Contributors. (2012). *Introduction to cloud computing* [Online]. Available: http://en.wikipedia.org/wiki/Introduction_to_cloud_computing (Accessed May 23, 2012).

18. Wikipedia Contributors. (2012). *Artificial neural network* [Online]. Available: http://en.wikipedia.org/wiki/Artificial_neural_network (Accessed May 23, 2012).

19. A. Algom. (2012). *Cloud computing pay-per-use for on-demand scalability* [Online]. Available: http://www.ogf.org/OGF25/materials/1500/AvnerAlgomIGT-OGF25.pdf (Accessed May 23, 2012).

20. Wikipedia Contributors. (2012). *Ammonium iron (II) sulfate* [Online]. Available: http://en.wikipedia.org/wiki/Ferrous_ammonium_sulphate (Accessed Jun. 6, 2012).

21. C.E. Housecroft, E.C. Constable, *Chemistry: An Introduction to Organic, Inorganic, and Physical Chemistry*, 3rd Ed. Upper Saddle River, NJ: Prentice Hall, 2005, pp. 349–353.

22. L. Rosenfeld, *Four Centuries of Clinical Chemistry*. Boca Raton, FL: CRC Press, 1999, pp. 255–258.

23. Wikipedia Contributors. (2012). *Colorimeter (chemistry)* [Online]. Available: http://en.wikipedia.org/wiki/Colorimeter_(chemistry)#cite_ref-2 (Accessed Jun. 6, 2012).

24. HunterLab. (2008). *Colorimeters versus spectrophotometers* [Online]. Available: http://www.hunterlab.com/appnotes/an03_95r.pdf (Accessed June 13, 2012).

25. Wikipedia Contributors. (2012). *Colorimetry (chemical method)* [Online]. Available: http://en.wikipedia.org/wiki/Colorimetry_(chemical_method)#cite_ref-1 (Accessed Jun. 13, 2012).

26. C. Burgess, "The Basics of Spectrophotometric Measurement," in *UV-Visible Spectrophotometry of Water and Wastewater* (*Techniques and Instrumentation in Analytical Chemistry*), vol. 27, O. Thomas and C. Burgess, Eds. Amsterdam, Netherland: Elsevier, 2007, pp. 1-19.

27. Wikipedia Contributors. (2013). *Color Vision* [Online]. Available: http://en.wikipedia.org/wiki/Color_vision (Accessed Feb. 11, 2013).

28. Wikipedia Contributors. (2012). *Charge-coupled device* [Online]. Available: http://en.wikipedia.org/wiki/Charge-coupled_device (Accessed Jul. 2, 2012).

29. S. Fogelman, H. Zhao, and M. Blumenstein, "A rapid analytical method for predicting the oxygen demand of wastewater," *Anal Bioanal Chem*, vol. 386, no. 6, pp. 1773–1779, Nov. 2006.

30. O. Thomas and F. Theraulaz, "Aggregate Organic Constituents," in *UV-Visible Spectrophotometry of Water and Wastewater* (*Techniques and Instrumentation in Analytical Chemistry*), vol. 27, O. Thomas and C. Burgess, Eds. Amsterdam, Netherland: Elsevier, 2007, pp. 89-114.

31. A. Charef, A. Ghauch, P. Baussand and M. Martin-Bouyer, "Water quality monitoring using a smart sensing system," *Measurement*, vol. 28, no. 8, pp. 219-224, Oct. 2000.

32. G. Langergraber, N. Fleischmann, F. Hofstaedter, and A. Weingartner, "Monitoring of a paper mill wastewater treatment plant using UV/VIS spectroscopy," *Water Science and Technology*, vol. 49, no. 1, pp. 9-14, 2004.

33. M.N. Pons, S.L. Bonte, O. Potier, "Spectral analysis and fingerprinting for biomedia characterization," *Journal of Biotechnology*, vol. 113, no. 1-3, pp. 211-230, Sep. 2004.

34. N. Fleischmann, G. Langergraber, *et al. On-line and in-situ measurement of turbidity and COD in wastewater using UV/VIS spectrometry* [Online]. Available: http://www.s-can.at/medialibrary/publications/p_2001_06.pdf (Accessed Mar. 26, 2012).

35. L. Cherta, J. Beltran, and T. Portoles, “Multiclass determination of 66 organic micropollutants in environment water samples by fast gas chromatography-mass spectrometry¸” *Anal Bioanal Chem*, vol. 402, no. 7, pp. 2301–2314, Sep. 2012.

36. S. Fogelman, H. Zhao, and M. Blumenstein, “A rapid analytical method for predicting oxygen demand of wastewater,” *Anal Bioanal Chem*, vol 386, no. 6, pp. 1773-1779, Oct. 2006.

37. S. Fogelman, H. Zhao, and M. Blumenstein, “Estimation of chemical oxygen demand by ultraviolate spectroscopic profiling and artificial neural networks,” *Neural Comput & Applic*, vol 15, no. 3-4, pp. 197-203, Jun. 2006.

38. N. Benjathapanun, W.J.O. Boyle, and K.T.V. Grattan, "Binary encoded 2nd-differential spectrometry using UV-Vis spectral data and neural networks in the estimation of species type and concentration," *Science, Measurement, and Technology, Proc. IEEE*, vol.144, no.2, pp.73,80, Mar. 1997
doi: 10.1049/ip-smt:19970713

39. F.M. Ham, G.M. Cohen, and C. Byoungho, "Neural network-based real-time detection of glucose using a non-chemical optical sensor approach," *Engineering in Medicine and Biology Society, 1990, Proc. IEEE* , vol., no., pp.480,482, 1-4 Nov. 1990. doi: 10.1109/IEMBS.1990.691176

40. M.A. Yongwen *et al.*, “Hybrid artificial neural network genetic algorithm technique for modeling chemical oxygen demand removal in anoxic/oxic process¸” *J Environ Sci Health A Tox Hazard Subst Environ Eng.*, vol. 46, no. 6, pp. 574-580, 2011.

41. H. Filho *et al.*, “A strategy for selecting calibration samples for multivariate modeling,” *Chemometr Intell Lab*, vol. 72, no. 1, pp. 83–91, Jun. 2004.

42. J. Dahlen *et al.*, “Determination of nitrate and other water quality parameters in ground water from UV/Vis spectra employing partial least squares regression,” *Chemosphere*, vol. 40, no. 1, pp. 71–77, Jan. 2000.

43. H. Khorassani *et al.*, “A simple UV spectrophotometric procedure for the survey of industrial sewage system,” *Wat Sci Tech*, vol. 39, no. 10-11, pp. 77–82, 1999.

44. W. Bourgeois, J. Burgess, and R. Stuetz, “On-line monitoring of wastewater quality: a review,” *J Chem Technol Biot*, vol. 76, no. 4, pp. 337–348, Mar. 2001.

45. A. Charef *et al.*, “Water quality monitoring using a smart sensing system,” *Measurement*, vol. 28, no. 3, pp. 219–224, Oct. 2000.

46. W. McCullock and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Math. Biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

47. F. Rosenblatt, "The perceptron: A probalistic model for information storage and organization in the brain," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

48. M. Minsky and S. Papert (1969), *Perceptron: An Introduction to Computational Geometry*. Cambridge, MA: MIT Press, 1969.

49. K. Fukushima, "Cognitron: A self-organizing multilayered neural network," *Biological Cybernetics*, vol. 20, no. 3–4, pp. 121–136, 1975.

50. Wikipedia Contributors. (2013). *Neural network* [Online]. Available: http://en.wikipedia.org/wiki/Neural_network#cite_ref-Kun1_12-0 (Accessed Feb. 11, 2013).

51. FriendlyARM. (2013). *Tiny6410 | S3C6410 ARM11 Stamp Module* [Online]. Available: http://www.friendlyarm.net/products/Tiny6410 (Accessed Feb. 11, 2013).

52. ARM. (2013). *S3C6410 ARM11 Mobile Processor by Samsung Electronics Co., Ltd* [Online]. Available: http://www.arm.com/community/partners/display_product/rw/ProductId/2644/ (Accessed Feb. 11, 2013).

53. Ocean Optics. (2013). *USB650 Red Tide Spectrometer* [Online]. Available: http://www.oceanoptics.com/Products/usb650.asp (Accessed Feb. 11, 2013).

54. ORACLE. (2013). *VirtualBox* [Online]. Available: https://www.VirtualBox.org/ (Accessed Feb. 11, 2013).

55. Canonical Ltd. (2013). *Download UBUNTU Server* [Online], Available: http://www.ubuntu.com/download/server (Accessed Feb. 11, 2013).

56. "Tera Term" Project Contributors. (2013). *Tera Term Home Page* [Online]. Available: http://ttssh2.sourceforge.jp/ (Accessed Feb. 11, 2013).

57. OceanOptics. (2013). *Spectral Software Made Easy* [Online]. Available: http://www.oceanoptics.com/Products/spectrasuite.asp (Accessed Feb. 11, 2013).

58. Netbeans Project Contributors. (2013). *Netbeans IDE* [Online]. Available: http://netbeans.org/ (Accessed Feb. 11, 2013).

59. A. Shaw, “Cloud computing enabled portable smart spectrophotometer,” M.S. thesis, CMPE, SJSU, San Jose, CA, 2011.

60. S. Nissen *et al*. *Fast artificial neural network library* [Online]. Available: http://leenissen.dk/fann/wp/ (Accessed Feb. 12, 2012).

61. Webopedia Contributors. (2013). Cloud-based [Online]. Available: http://www.webopedia.com/TERM/C/cloud_based.html (Accessed Feb, 24, 2013).

62. W. Boyles, “The Science of Chemical Oxygen Demand,” Hach Co., Loveland, CO, Technical Information Series, Booklet No. 9, 1997.

63. Canonical Ltd. (2013). *UBUNTU Cloud infrastructure* [Online]. Available: https://help.ubuntu.com/community/UbuntuCloudInfrastructure (Accessed Feb. 24, 2013).

64. OpenStack Project Contributors. (2013). *Open source software for building private and public clouds* [Online]. Available: http://www.openstack.org/ (Accessed Feb. 24, 2013).

65. K. Jackson. (Feb. 17, 2011). *Running OpenStack under VirtualBox – A Complete Guide (Part 1)* [Online]. Available: http://uksysadmin.wordpress.com/2011/02/17/running-openstack-under-VirtualBox-a-complete-guide/ (Accessed Feb. 24, 2013).

66. K. Jackson. (Feb. 24, 2011). *Running OpenStack under VirtualBox – A Complete Guide (Part 2)* [Online]. Available: http://uksysadmin.wordpress.com/2011/02/24/running-openstack-under-VirtualBox-a-complete-guide-part-2/ (Accessed Feb. 24, 2013).

67. Canonical Ltd. (2013). *Open SSH Server* [Online]. Available: https://help.ubuntu.com/10.04/serverguide/openssh-server.html (Accessed Sep. 9, 2012).

68. Canonical Ltd. (2013). *SettingUpNFSHowTo* [Online]. Available: https://help.ubuntu.com/community/SettingUpNFSHowTo (Accessed Sep. 9, 2012).

69. M. Johnston. (2012). *Dropbear SSH* [Online]. Available: http://matt.ucc.asn.au/dropbear/dropbear.html (Accessed Sep. 9, 2012).

70. E. Andersen. (1999). *Busybox* [Online]. Available: http://www.busybox.net/ (Accessed Sep. 9, 2012).

71. J. York. (Apr. 8, 2009). *Using Public Keys With Dropbear SSH Client* [Online]. Available: http://yorkspace.wordpress.com/2009/04/08/using-public-keys-with-dropbear-ssh-client/ (Accessed Sep. 9, 2012).

72. Spectra Services. (2012). *DT-MINI-2 Deuterium Tungsten Halogen Light Source* [Online]. Available: http://spectraservices.com/Merchant2/merchant.mvc?Screen=PROD&Product_Code=DTMINI2 (Accessed on Mar. 23, 2012).

APPENDIX A

ARM11 Board Bring Up with A Simple Device Driver

I. INTRODUCTION

The Tiny6410 is a development board from FriendlyARM [51] using a SMDK6410 platform from Samsung. The processor on this board is an ARM11 implementation from the Samsung Incorporation (http://www.samsung.com/us).

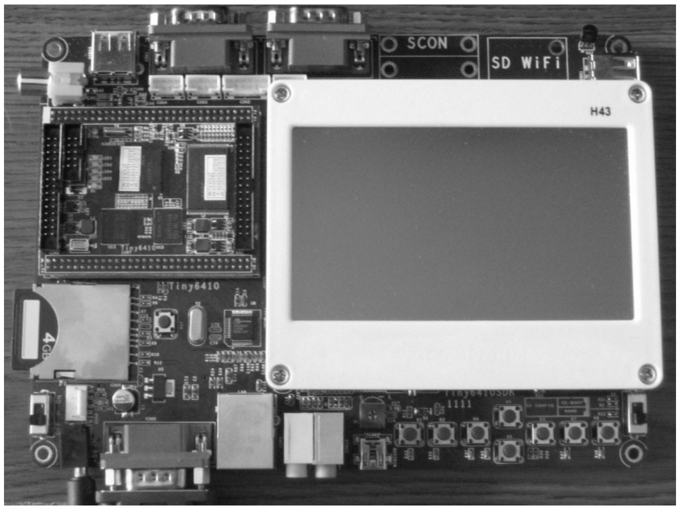


Fig. 32. Tiny6410 SDK

This paper discusses the development environment setup for this board to perform experiments with Linux and device drivers. This paper is divided into three major sections to discuss the development environment setup, bringing up the Tiny6410 board, and writing a simple device driver. The first section describes the tools and environment setup. The second section describes how to compile and start Linux on Tiny6410. The third section describes an implementation of a simple dynamically loadable device driver.

II. Development Environment Setup

A. Base environment

The host computer was running Windows 7 Enterprise Edition. It had the ORACLE VirtualBox application running to emulate UBUNTU as a guest operating system.

```
kpatra-VirtualBox:~> uname -a
Linux kpatra-VirtualBox 3.0.0-17-generic #30-Ubuntu SMP Thu Mar 8
17:34:21 UTC 2012 i686 i686 i386 GNU/Linux
```

The guest OS used in this experiment had a mount point of the Windows file system to share binaries to be installed and executed on the ARM11 board. The shell environment in Guest-OS was "tcsh."

B. Cross compiler installation for ARM11

The SD card delivered as part of the package contained all the required tools for cross compilation and the source code for Linux. The cross compiler tool set was shipped as arm-linux-gcc-4.5.1-v6-vfp-20101103.tar.gz. Once installed using gunzip and tar command, it created an "opt" directory under the run directory of the command. In this experiment setup, the tool chain was installed at $PROJ_ROOT/mini6410/Linux directory (which could be any independent location, depending on user setup).

```
$> cd $PROJ_ROOT/mini6410/Linux
$> cp $TINY6410_PKG_DIR/ arm-linux-gcc-4.5.1-v6-vfp-20101103.tar.gz .
$> gunzip arm-linux-gcc-4.5.1-v6-vfp-20101103.tar.gz
$> tar –xvf arm-linux-gcc-4.5.1-v6-vfp-20101103.tar
```

Once installed, the tool chain could be found at ./opt/FriendlyARM/toolschain /4.5.1/bin/ directory. To make this tool chain available by default, the following line was added into the ~/.tcshrc file.

```
setenv PATH
${PROJ_ROOT}/mini6410/Linux/opt/FriendlyARM/toolschain/4.5.1/bin:${PATH}
```

To test the installation, the following command should run on a new terminal.

```
$> arm-Linux-cc -v
Using built-in specs.
COLLECT_GCC=arm-Linux-cc
.   .   .
Thread model: posix
gcc version 4.5.1 (ctng-1.8.1-FA)
```

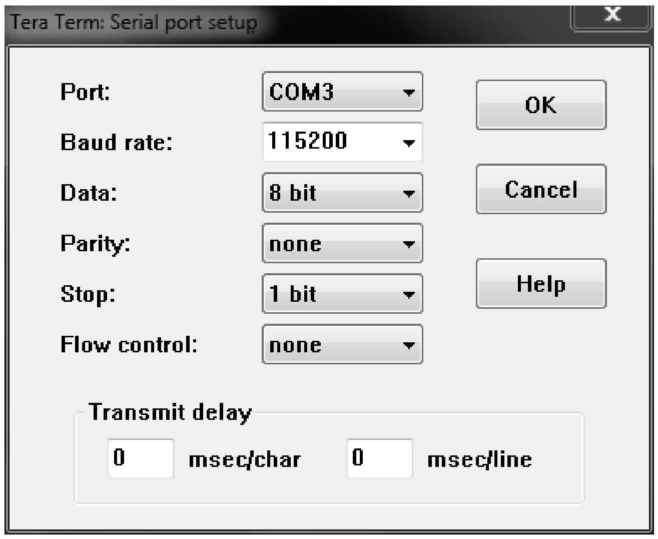
C. Linux source code installation

The Linux source code was shipped as the linux-2.6.38-20110718.tar.gz file. This source can be installed in the same way as the cross compiler installation. This installs all of the source code into ./linux-2.6.38 directory.

```
$> cd $PROJ_ROOT/Tiny6410/Linux
$> cp $MINI6410_PKG_DIR/ linux-2.6.38-20110718.tar.gz .
$> gunzip linux-2.6.38-20110718.tar.gz
$> tar –xvf linux-2.6.38-20110718.tar
```

D. HyperTerminal installation

The Windows 7 host did not have a HyperTerminal for serial communication monitoring. There is an open source hyper terminal "Tera Term" [56], which is very good for serial communication monitoring and can be used as a console for Tiny6410 SDK. In addition, the "Tera Term" application is a true multipurpose terminal application supporting both serial and TCP/IP-based communication (like telnet or SSH).


Tera Term: Serial port setup

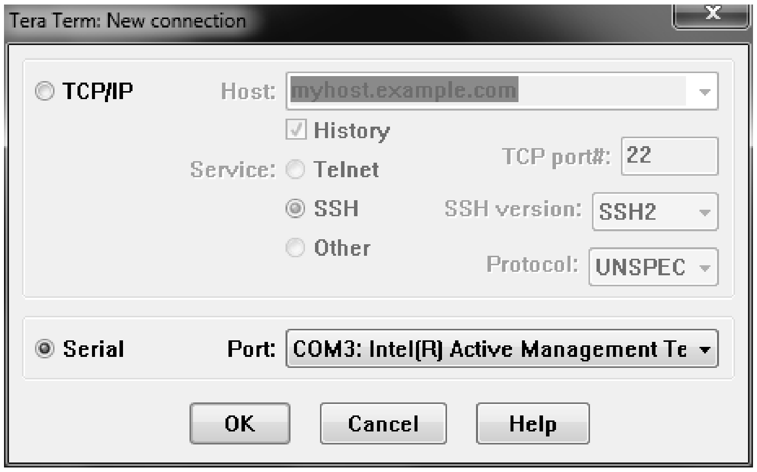
Port:	COM3
Baud rate:	115200
Data:	8 bit
Parity:	none
Stop:	1 bit
Flow control:	none

Transmit delay

0 msec/char 0 msec/line

Fig. 33. "Tera Term" Setup for Serial Port

Fig. 33 shows the application menu for the setup needed to use "Tera Term" as a console for Tiny6410. This menu can be opened using the "menu-Setup-Serial Connection…" menu switch. In this experiment, the port "COM3" was used because it was auto assigned to the USB-to-serial port converter attached to the host computer system. The baud rate was set to 115200 for a data width of 8 bit with no parity and 1 stop bit. Flow control was set to 0 and the transmit delay was set to 0 for both char and line.


Tera Term: New connection

TCP/IP Host: myhost.example.com

[✓ History]

Service: Telnet [✓ SSH] Other

TCP port#: 22

SSH version: SSH2

Protocol: UNSPEC

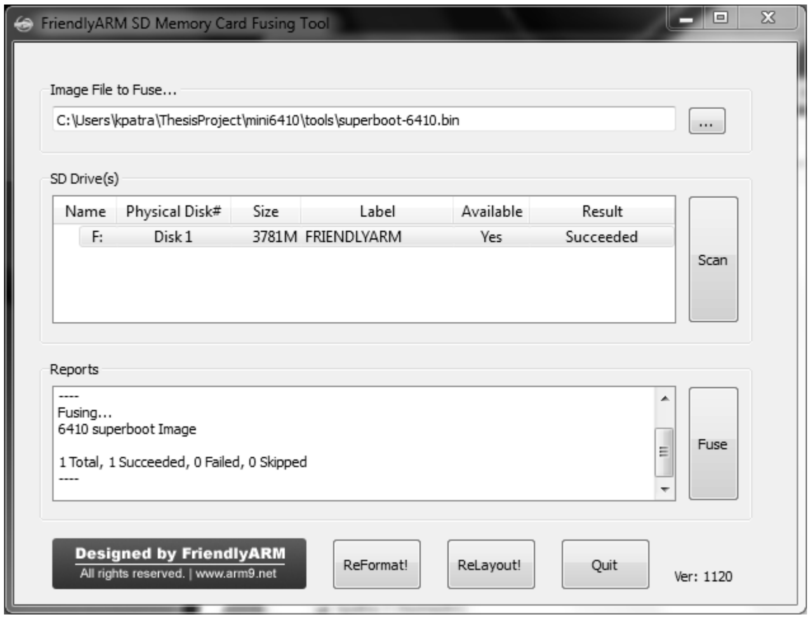
[✓ Serial] Port: COM3: Intel(R) Active Management Te

Fig. 34. New Connection Menu of Tera Term Application

To start a communication, the menu "File-New Connection" was used. Fig. 34 shows the serial port and port number that were selected to start a new communication.

E. SD card preparation with BIOS

To boot the Tiny6410 from the SD card, the SD card must be prepared beforehand. The tool package for Tiny6410 contained the boot loader superboot-6410.bin in the tools directory. This boot loader was used to boot the system. To prepare the SD card, the program SD-Flasher.exe in the Tools folder was supplied in the software package. This program had to be executed using administrator privileges ("Run as administrator" can be found with a right click on this executable file). This brought up a GUI as shown in Fig. 35.



C:\Users\kpatra\ThesisProject\mini6410\tools\superboot-6410.bin

Name	Physical Disk#	Size	Label	Available	Result
F:	Disk 1	3781M	FRIENDLYARM	Yes	Succeeded

```
----
Fusing...
6410 superboot Image

1 Total, 1 Succeeded, 0 Failed, 0 Skipped
----
```

Fig. 35. SD-Flasher.exe GUI

Superboot-6410.bin was selected as the image file to fuse. Once the SD card was mounted on the host, the scan option showed the mounted SD card. Once the Fuse button was clicked, the program fused the BIOS into the SD card. The status report was shown in the Reports section of the GUI. In case the SD card is not shown as available ("No" is displayed in the Available column of the SD drives), it can be made available using the ReLayout button).

III. Bringing Up the Tiny6410 Board

A. Compiling Linux for Tiny6410

There were two steps to build a Linux image named zImage. The first step was to copy the Tiny6410 n43 configuration and then build the zImage using Make. The following commands were used to build the Linux image. Copying the configuration is a one-time job; once done, it need not be repeated for every build of the zImage.

```
$> cd $PROJ_ROOT/mini6410/Linux/linux-2.6.38/
$> cp config_mini6410_n43 .config
$> make zImage
```

B. *Preparing the SD card for Linux installation*

A specific directory structure and a couple of files in addition to the compiled zImage were needed to install the compiled Linux on Tiny6410 SDK. A directory named "images" was created at the top-level directory $SD_ROOT of the SD card and two files—"FriendlyARM.ini" and "superboot-6410.bin" —were copied from Tiny6410/tools/ in this "images" directory.

```
$> mkdir $SD_ROOT/images
$> cd $SD_ROOT/images
$> cp $PROJ_ROOT/mini6410/tools/FriendlyARM.ini  .
$> cp $PROJ_ROOT/mini6410/tools/superboot-6410.bin  .
```

The disk image of a root file system was downloaded from the download section of the FriendlyARM website [1]. The name of the root file system was rootfs_qtopia_qt4_20110305.img. This root file system image file was copied to the $SD_ROOT/Linux dir.

```
$> mkdir $SD_ROOT/images/Linux
$> cp rootfs_qtopia_qt4_20110305.img  \
   $SD_ROOT/images/Linux
```

Next the compiled zImage was copied to the $SD_ROOT/images/Linux directory. In the following command example, the LNX_COMP_DIR was set to $PROJ_ROOT/Tiny6410/Linux/linux-2.6.38 dir.

```
$> cp $LNX_COMP_DIR/ arch/arm/boot/zImage $SD_ROOT/images/Linux
```

The content of the FriendlyARM.ini file included the following:

```
Action=install
OS= Linux
Linux-BootLoader = superboot-6410.bin
Linux-Kernel = Linux/zImage
Linux-CommandLine = root=/dev/mtdblock2 rootfstype=yaffs2 init=/linuxrc
console=ttySAC0,115200
Linux-RootFs-InstallImage = Linux/rootfs_qtopia_qt4_20110305.img
Linux-RootFs-RunImage = Linux/ rootfs_qtopia_qt4_20110305.img
```

Table 5 explains each of the file parameters.

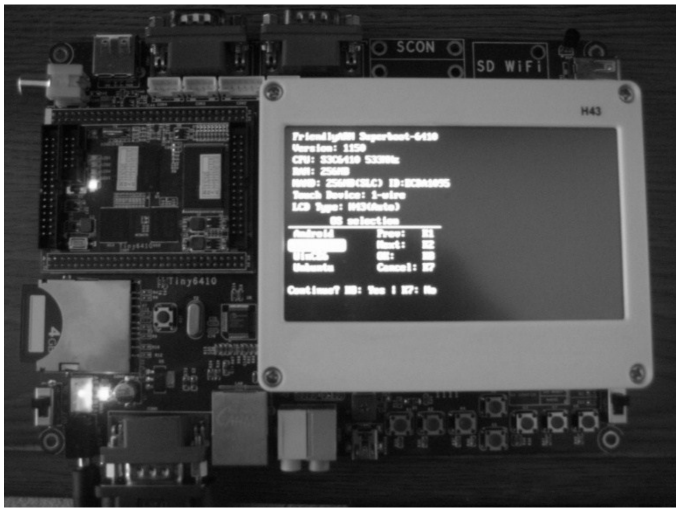
Table 5. Content of FriendlyARM.ini File

Parameter names	Details
Action	This can be either "install" or "run." In install mode, the boot loader installs the OS in Tiny6410's flash memory. In "run" mode, the compiled OS is directly run from the SD card.
OS	The type of OS can be anything as long as the derived parameter names are present in the file. The derived parameters are • <OS>-BootLoader • <OS>-Kernel • <OS>-CommandLine • <OS>-RootFs-InstallImage • <OS>-RootFs-RunImage

<OS>-BootLoader	The boot loader executable
<OS>-Kernel	The kernel image file
<OS>-CommandLine	The command line for system boot up
<OS>-RootFs-InstallImage	The file system image files for installation
<OS>-RootFs-RunImage	The file system image files for direct run

C. Installing Linux from SD Card

Once the SD card was prepared, it was inserted into the SD card slot of the Tiny6410. To start the installation process, the switch S2 was put into SDBOOT position and the power switch S1 was turned on.

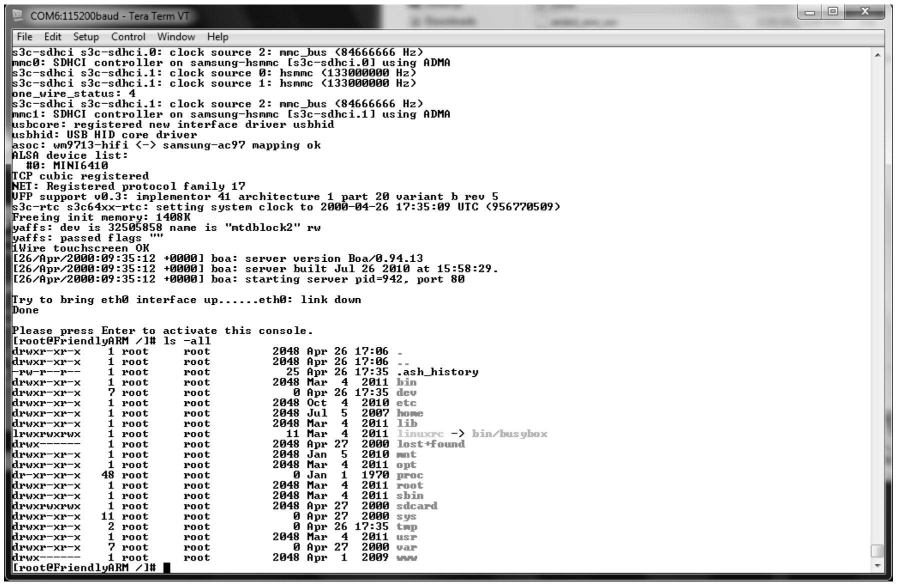


```
FriendlyARM Superboot-6410
Version: 1150
CPU: S3C6410 533MHz
RAM: 256MB
NAND: 256MB(SLC) ID:ECDA1095
Touch Device: 1-wire
LCD Type: H43(Auto)
       OS selection
Android          Prev:    K1
                 Next:    K2
WinCE            OK:      K8
Ubuntu           Cancel:  K7

Continue? K8: Yes | K7: No
```

Fig. 36. Installing the OS from the SD Card

The boot menu appeared on the LCD. Using K1 (up), K2 (down), and K8 (OK) the Linux system was selected. Once the system was selected, the boot loader began to write the kernel image and root file system in the Tiny6410 flash memory.


COM6:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
s3c-sdhci s3c-sdhci.0: clock source 2: mmc_bus (84666666 Hz)
mmc0: SDHCI controller on samsung-hsmmc [s3c-sdhci.0] using ADMA
s3c-sdhci s3c-sdhci.1: clock source 0: hsmmc (133000000 Hz)
s3c-sdhci s3c-sdhci.1: clock source 1: hsmmc (133000000 Hz)
one_wire_status: 4
s3c-sdhci s3c-sdhci.1: clock source 2: mmc_bus (84666666 Hz)
mmc1: SDHCI controller on samsung-hsmmc [s3c-sdhci.1] using ADMA
usbcore: registered new interface driver usbhid
usbhid: USB HID core driver
asoc: wm9713-hifi <-> samsung-ac97 mapping ok
ALSA device list:
  #0: MINI6410
TCP cubic registered
NET: Registered protocol family 17
VFP support v0.3: implementor 41 architecture 1 part 20 variant b rev 5
s3c-rtc s3c64xx-rtc: setting system clock to 2000-04-26 17:35:09 UTC (956770509)
Freeing init memory: 1408K
yaffs: dev is 32505858 name is "mtdblock2" rw
yaffs: passed flags ""
1Wire touchscreen OK
[26/Apr/2000:09:35:12 +0000] boa: server version Boa/0.94.13
[26/Apr/2000:09:35:12 +0000] boa: server built Jul 26 2010 at 15:58:29.
[26/Apr/2000:09:35:12 +0000] boa: starting server pid=942, port 80

Try to bring eth0 interface up......eth0: link down
Done

Please press Enter to activate this console.
[root@FriendlyARM /]# ls -all
drwxr-xr-x    1 root     root          2048 Apr 26 17:06 .
drwxr-xr-x    1 root     root          2048 Apr 26 17:06 ..
-rw-r--r--    1 root     root            25 Apr 26 17:35 .ash_history
drwxr-xr-x    1 root     root          2048 Mar  4  2011 bin
drwxr-xr-x    7 root     root             0 Apr 26 17:35 dev
drwxr-xr-x    1 root     root          2048 Oct  4  2010 etc
drwxr-xr-x    1 root     root          2048 Jul  5  2007 home
drwxr-xr-x    1 root     root          2048 Mar  4  2011 lib
lrwxrwxrwx    1 root     root            11 Mar  4  2011 linuxrc -> bin/busybox
drwx------    1 root     root          2048 Apr 27  2000 lost+found
drwxr-xr-x    1 root     root          2048 Jan  5  2010 mnt
drwxr-xr-x    1 root     root          2048 Mar  4  2011 opt
dr-xr-xr-x   48 root     root             0 Jan  1  1970 proc
drwxr-xr-x    1 root     root          2048 Mar  4  2011 root
drwxr-xr-x    1 root     root          2048 Mar  4  2011 sbin
drwxrwxrwx    1 root     root          2048 Apr 27  2000 sdcard
drwxr-xr-x   11 root     root             0 Apr 27  2000 sys
drwxr-xr-x    2 root     root             0 Apr 26 17:35 tmp
drwxr-xr-x    1 root     root          2048 Mar  4  2011 usr
drwxr-xr-x    7 root     root             0 Apr 27  2000 var
drwx------    1 root     root          2048 Apr  1  2009 www
[root@FriendlyARM /]#
```

Fig. 37. Linux Boot Up Console on Tera Term

Once the boot loader was done with flash writing, the system was restarted by putting the S2 in flash booting position and toggling the power switch S1. The system booted and the boot message could be captured at the “Tera Term” console as seen in Fig. 37.

D. Testing the installation

There are seven example programs given in the $PROJ_ROOT/mini6410/Linux /examples directory. However, the pwm test is a good test to verify the installation. Inside the pwm directory, there are two files—Makefile and pwm_test.c. The first step is to clean and rebuild this test program as follows:

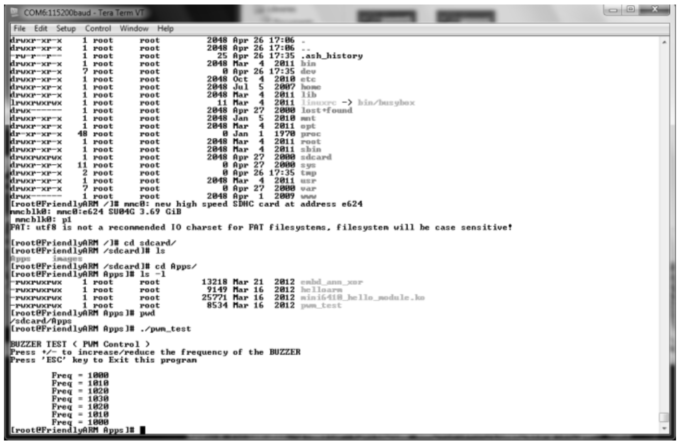
```
$> cd $PROJ_ROOT/mini6410/Linux/examples/pwm
$> make clean
$> make
```

This created an ARM11 executable named pwm_test, which was copied into the $SD_ROOT/Apps dir. After booting the Linux on the Tiny6410 SDK, the SD card was inserted into the SD card slot and the program was invoked from the /sdcard/Apps directory.

The following commands were executed on SDK using the “Tera Term” console.

```
$> cd /sdcard/Apps
$> ./pwm_test
```

The console output is shown in Fig. 38.



```
drwxr-xr-x    1 root     root         2048 Apr 26 17:06 .
drwxr-xr-x    1 root     root         2048 Apr 26 17:06 ..
-rw-r--r--    1 root     root           25 Apr 26 17:35 .ash_history
drwxr-xr-x    1 root     root         2048 Mar  4  2011 bin
drwxr-xr-x    7 root     root            0 Apr 26 17:35 dev
drwxr-xr-x    1 root     root         2048 Oct  4  2010 etc
drwxr-xr-x    1 root     root         2048 Jul  5  2007 home
drwxr-xr-x    1 root     root         2048 Mar  4  2011 lib
lrwxrwxrwx    1 root     root           11 Mar  4  2011 linuxrc -> bin/busybox
drwx------    1 root     root         2048 Apr 27  2000 lost+found
drwxr-xr-x    1 root     root         2048 Jan  5  2010 mnt
drwxr-xr-x    1 root     root         2048 Mar  4  2011 opt
dr-xr-xr-x   48 root     root            0 Jan  1  1970 proc
drwxr-xr-x    1 root     root         2048 Mar  4  2011 root
drwxr-xr-x    1 root     root         2048 Mar  4  2011 sbin
drwxrwxrwx    1 root     root         2048 Apr 27  2000 sdcard
drwxr-xr-x   11 root     root            0 Apr 27  2000 sys
drwxr-xr-x    2 root     root            0 Apr 26 17:35 tmp
drwxr-xr-x    1 root     root         2048 Mar  4  2011 usr
drwxr-xr-x    7 root     root            0 Apr 27  2000 var
drwx------    1 root     root         2048 Apr  1  2009 www
[root@FriendlyARM /]# mmc0: new high speed SDHC card at address e624
mmcblk0: mmc0:e624 SU04G 3.69 GiB
 mmcblk0: p1
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem will be case sensitive!

[root@FriendlyARM /]# cd sdcard/
[root@FriendlyARM /sdcard]# ls
Apps    images
[root@FriendlyARM /sdcard]# cd Apps/
[root@FriendlyARM Apps]# ls -l
-rwxrwxrwx    1 root     root        13218 Mar 21  2012 embd_ann_xor
-rwxrwxrwx    1 root     root         9149 Mar 16  2012 helloarm
-rwxrwxrwx    1 root     root        25771 Mar 16  2012 mini6410_hello_module.ko
-rwxrwxrwx    1 root     root         8534 Mar 16  2012 pwm_test
[root@FriendlyARM Apps]# pwd
/sdcard/Apps
[root@FriendlyARM Apps]# ./pwm_test

BUZZER TEST ( PWM Control )
Press +/- to increase/reduce the frequency of the BUZZER
Press 'ESC' key to Exit this program

        Freq = 1000
        Freq = 1010
        Freq = 1020
        Freq = 1030
        Freq = 1020
        Freq = 1010
        Freq = 1000
[root@FriendlyARM Apps]#
```

Fig. 38. pwm_test Run using the “Tera Term” Console

This program uses a buzzer and applies different frequencies of sound. Users may increase or decrease the frequency of the sound using the “+” or “–” keys. To terminate this program, the “Esc” key must be used.

IV. Writing Simple Device Driver

A. Writing the sample device driver

The simplest device driver was given in the Linux source code as Tiny6410_hello_module.c. This simple loadable device driver prints a message on the console. The source listing is shown in Table 6.

Table 6. Source Code Listing for the Tiny6410_hello_module.c

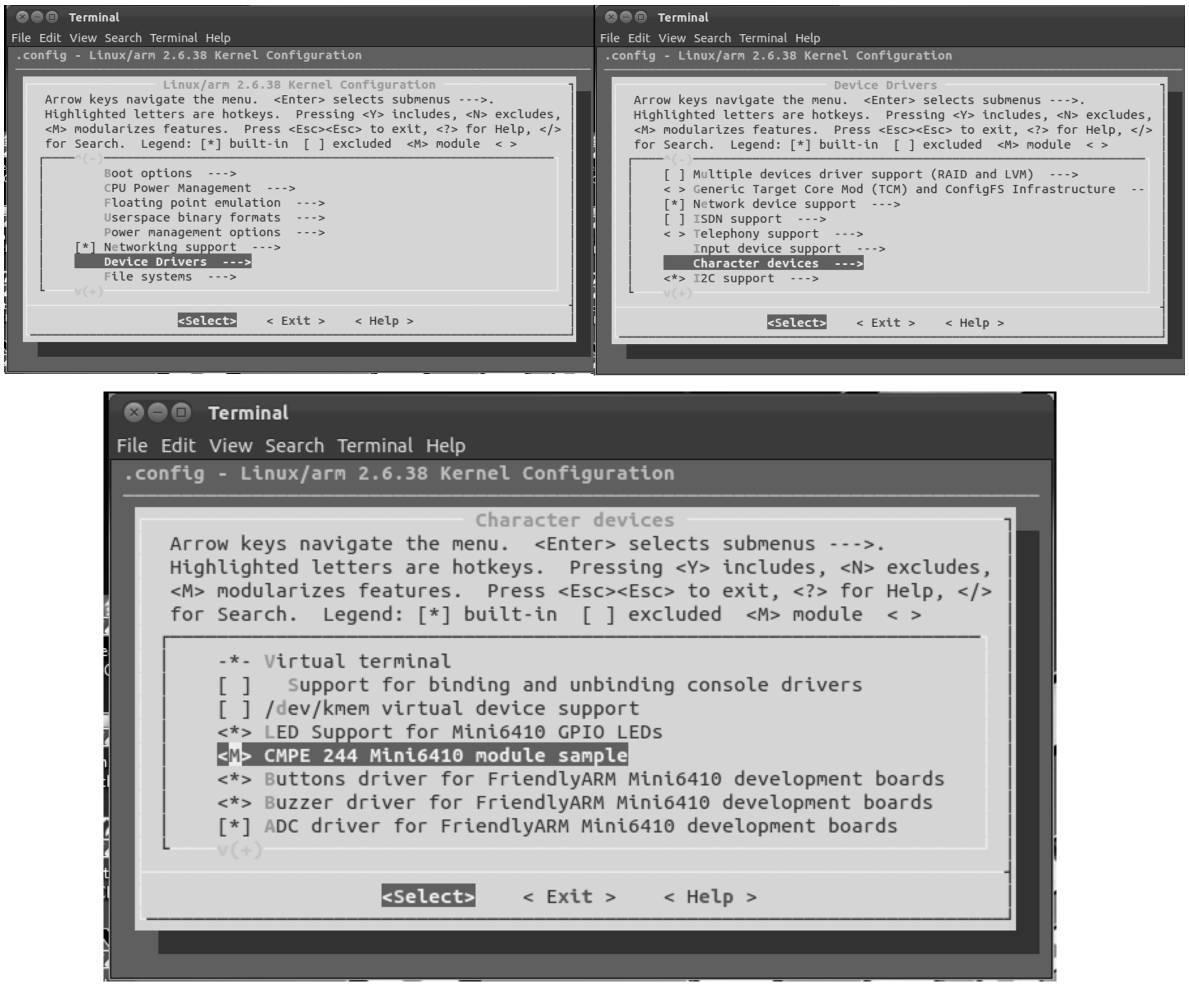
Line #	Source code
1	`#include <Linux/kernel.h>`
2	`#include <Linux/module.h>`
3	`static int __init`
	` Tiny6410_hello_module_init(void){`
4	`  printk(" Kaushik CMPE244 Hello, Tiny6410 module is installed !\n");`
5	`  return 0;}`
6	`static void __exit`
	` Tiny6410_hello_module_cleanup(void){`
7	`  printk(" Kaushik CMPE 244 Good-bye, Tiny6410 module was removed!\n");}`
8	`module_init(Tiny6410_hello_module_init);`
9	`module_exit(Tiny6410_hello_module_cleanup);`
10	`MODULE_LICENSE("GPL");`

Lines 1–2 include the required header files Linux/kernel.h and Linux/module.h. Lines 3–5 define the initialization function for this module (Line 4 prints the message on the console). Lines 6–7 define the exit process for the module. Lines 8–9 register the module's initialization and exit functions into the OS. Line 10 defines the license type.

B. Adding module in the config menu

To add this module into the menuconfig, the following section needs to be added in the Kconfig file at linux-2.6.38/drivers/char directory.

```
config TINY6410_HELLO_MODULE
  tristate "CMPE 244 Tiny6410 module sample"
  depends on CPU_S3C6410
  help
     CMPE244 Tiny6410 module sample.
```



```
.config - Linux/arm 2.6.38 Kernel Configuration

                 Linux/arm 2.6.38 Kernel Configuration
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
     ^(-)
         Boot options  --->
         CPU Power Management  --->
         Floating point emulation  --->
         Userspace binary formats  --->
         Power management options  --->
     [*] Networking support  --->
         Device Drivers  --->
         File systems  --->
     v(+)

            <Select>    < Exit >    < Help >
```

```
.config - Linux/arm 2.6.38 Kernel Configuration

                          Device Drivers
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >
     ^(-)
     [ ] Multiple devices driver support (RAID and LVM)  --->
     < > Generic Target Core Mod (TCM) and ConfigFS Infrastructure  --
     [*] Network device support  --->
     [ ] ISDN support  --->
     < > Telephony support  --->
         Input device support  --->
         Character devices  --->
     <*> I2C support  --->
     v(+)

            <Select>    < Exit >    < Help >
```

```
.config - Linux/arm 2.6.38 Kernel Configuration

                        Character devices
 Arrow keys navigate the menu.  <Enter> selects submenus --->.
 Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
 <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help, </>
 for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < >

     -*- Virtual terminal
     [ ]   Support for binding and unbinding console drivers
     [ ] /dev/kmem virtual device support
     <*> LED Support for Mini6410 GPIO LEDs
     <M> CMPE 244 Mini6410 module sample
     <*> Buttons driver for FriendlyARM Mini6410 development boards
     <*> Buzzer driver for FriendlyARM Mini6410 development boards
     [*] ADC driver for FriendlyARM Mini6410 development boards
     v(+)

            <Select>    < Exit >    < Help >
```

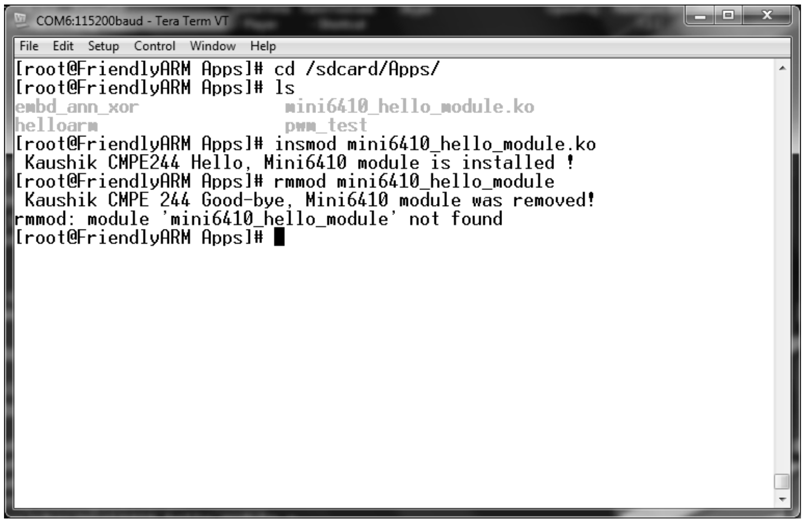
Fig. 39. Menu Configuration Selection

C. Compiling "hello world device driver."

The "hello world device driver" was selected as a loadable module using "make menuconfig" at the linux-2.6.38 directory. Through a series of selection operations as following Fig. 39, the "hello world device driver" was selected as a loadable module. Once the configuration was saved, the modules were built using "make modules" at the linux-2.6.38 directory. After the build process was done, the Tiny6410_hello_module.ko was generated at the drivers/char directory. This Tiny6410_hello_module.ko was then copied into $SD_ROOT/Apps dir. Then the driver was tested using the Tiny6410 SDK with the following commands:

```
$> cd /sdcard/Apps
$> insmod Tiny6410_hello_module.ko
$> rmmod Tiny6410_hello_module.ko
```

Fig. 40 shows the output on the "Tera Term" console on this example run.


COM6:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
[root@FriendlyARM Apps]# cd /sdcard/Apps/
[root@FriendlyARM Apps]# ls
embd_ann_xor              mini6410_hello_module.ko
helloarm                  pwm_test
[root@FriendlyARM Apps]# insmod mini6410_hello_module.ko
 Kaushik CMPE244 Hello, Mini6410 module is installed !
[root@FriendlyARM Apps]# rmmod mini6410_hello_module
 Kaushik CMPE 244 Good-bye, Mini6410 module was removed!
rmmod: module 'mini6410_hello_module' not found
[root@FriendlyARM Apps]# 
```

Fig. 40. Testing Sample Loadable Device Driver

V. Conclusion

With all the experiments done with the Tiny6410 SDK, it was concluded that the development environment was right, the board was functioning correctly, and a dynamically loadable device driver could be implemented.

APPENDIX B

Setting Up UV Spectrometer SDK on ARM11

I. INTRODUCTION

The OceanOptics Corporation (http://www.oceanoptics.com) is the world's first company to bring miniature spectrometer. This company manufactures different types of spectrometers, of which the spectrometer with the USB interface is the most useful in a university laboratory environment. The advantage of this type of USB-interfaced spectrometer is having a very easy interface for data collection. In this experiment, Ocean Optics Red Tide USB650 Spectrometer [53] was used.

Fig. 41. RED TIDE USB650 Device

However, OceanOptics provides only a Windows-based interface application to collect data. This document explains the compilation and installation of SDK on an i386 and an ARM11-based platform running Linux operating system. In the following sections, download and installation instructions, as well as the software directory structure, are described. All the instructions documented in this paper are based on the following operating system: Ubuntu SMP Thu Mar 8 17:34:21 UTC 2012 i686 i686 i386 GNU/Linux. A tcsh shell was used for command-line input.

II. DOWNLOAD AND INSTALLATION

A. Downloading the package

OceanOptics usually does not provide the SDK source for any platform other than Windows. The Computer Engineering Department of San Jose State University obtained special permission from OceanOptics to get the SDK source code and compile it on a platform other than Windows. In the SDK source, there are two directories containing platform-specific source code as shown in Table 7.

Table 7. Spectrometer Source Code per Platform

Directory	Platform
OEM/SeaBreezeOSX	Apple Macintosh OSX operating system.
OEM/SeaBreezeWinSource	Microsoft Windows operating system.

Because Apple Macintosh OSX is a variation of the UNIX operating system, the corresponding source code was selected for compilation on a Linux platform for both i386 and ARM11 architecture. The i386 architecture was included in this experiment to eliminate operating system porting-related issues for the SDK. Additionally, it was assumed that the environment variable $SEA_BREEZE_DEV_ROOT contained the source code root for the SDK. For the current experiment, the value of this environment variable was as follows:

```
$> echo $SEA_BREEZE_DEV_ROOT
/import/thesis/Study/Thesis/COD-
ANN/SpectraSuite/SeaBreezeTiny6410/Linux_122011/OceanOptics/OceanOpticsF
TP/OEM/SeaBreezeOSX/seabreeze/
```

In addition, the libusb source code was downloaded from http://www.libusb.org because the ARM11-compatible libusb was not available in precompiled form. The downloaded package was installed in a directory named $SEA_BREEZE_DEV_ROOT, because this compiled libusb was compiled along with rest of the SDK. This experiment made libusb an integral part of OceanOptics SDK for USB spectrometer. In this experiment, the package for the source code of libusb was unpacked into the libusb directory.

```
$> ls $SEA_BREEZE_DEV_ROOT/libusb
libusb-1.0.8.tar.bz2
$> cd $SEA_BREEZE_DEV_ROOT/libusb
$> bunzip2 libusb-1.0.8.tar.bz2
$> ls
libusb-1.0.8.tar
$> tar –zvf libusb-1.0.8.tar
$> ls
libusb-1.0.8 libusb-1.0.8.tar
```

B. Installation

The top-level makefile was modified for the seabreeze SDK to implement a new set of targets. Additionally a new script was created to compile libusb on an ARM11 platform. There was no corresponding script to build libusb on an i386 platform because i386

UBUNTU already comes with a libusb package integrated. All the changes and additions are discussed in Section III.

Table 8. Top-Level Makefile Targets to Build/Test the Spectrometer Device Driver

Target	Action
default	To build seabreeze lib on Linux i386 in lib/i386_Linux directory.
test	To build the test program under the test/i386_Linux directory.
clean	To clean the temporary files.
clean_test	To clean the temporary files for testing.
install_test	To install the application along with the library in the test/install/i386_Linux directory. The sub directory "app" contains the executable and "lib" contains the required libraries.
run_test	To run the test application.
help	To display the list of targets and macros.

Table 8 shows the available make targets to build libraries and test applications. Table 9 shows the available macro to execute the targets for the specific architecture.

Table 9. List of Macros for Architecture to Build Spectrometer Device Driver

Macro	Values	Notes
TARGET	i386	This is the default value for the macro and compiles the application and libraries for i386 architecture.
	arm11	With this macro value, the application and libraries are compiled for ARM11 architecture.

To switch target architecture from i386 to ARM11 or vice versa, the "clean" target had to be run first. The following is the example command sequence used to build a test app for ARM11.

```
$> cd $SEA_BREEZE_DEV_ROOT
$> make clean TARGET=arm11
$> make clean_test TARGET=arm11
$> make TARGET=arm11
$> make install_test TARGET=arm11
```

The above set of commands was used to create the following directories under $SEA_BREEZE_DEV_ROOT/test dirs.

Table 10. Install Directory Structure for Spectrometer Device Driver

Directory	Notes
install/arm11_Linux/app	The test application executable is installed in this directory.
install/arm11_Linux/lib	All the required libraries are installed in this directory.

These installation directories (as shown in Table 10) can be copied into an ARM11-based platform and then the test application can be executed. For the i386 target architecture, the installation directory contained the platform directory i386_Linux.

III. CHANGES IN ORIGINAL PACKAGE

Three new files were created in the original extracted package content inside the $SEA_BREEZE_DEV_ROOT directory. Table 11 shows the files that were new or replaced with new revision.

Table 11. New/Changed Files to Build/Test Spectrometer Device Driver

File	Notes
Makefile	This is a replacement for the existing Makefile.
test/Makefile	This is a replacement for the existing Makefile.
libusb/libusb-1.0.8/local-install-arm11	This is a new script for compiling libusb for ARM11 architecture.

The following subsections describe each of the above files and their content.

A. The top-level makefile

The top-level makefile contained the top-level target and macro definitions.

```
export UNAME:=$(shell uname)
export TARGET=i386
export PLATFORM=$(TARGET)_$(UNAME)
export SEABREEZE=$(SEABREEZE_DIR)/include
export LIB_USB_DIR=$(SEABREEZE_DIR)/libusb/libusb-1.0.8
export LIB_USB_PATH=$(SEABREEZE_DIR)/libusb/libusb-1.0.8/install/lib/
export LIB_USB_INC_PATH=$(SEABREEZE_DIR)/libusb/libusb-
1.0.8/install/include/libusb-1.0
export SEABREEZE_LIB=$(SEABREEZE_DIR)/lib/$(PLATFORM)
export TCSH = /usr/bin/tcsh
export SEABREEZE_TEST=$(SEABREEZE_DIR)/test
```

The above macros defined different directories, paths, and shells.

```
export CC_PREFIX=
ifeq ($(TARGET),arm11)
export CC_PREFIX=arm-Linux-
endif
export CC=$(CC_PREFIX)gcc
export CPP=$(CC_PREFIX)g++
```

The above makefile statements defined the target compiler. Depending on the TARGET macro value, either gcc/g++ or arm-Linux-gcc/arm-Linux-g++ was used.

```
export CFLAGS=-c -Wall -Wunused -Wmissing-include-dirs -ggdb3 -
I${SEABREEZE} -fpic -fno-stack-protector
ifeq ($(TARGET),arm11)
CFLAGS += -I$(LIB_USB_INC_PATH)
endif
```

```
LFLAGS=-shared -Wl,--export-dynamic –nostdlib
ifeq ($(TARGET),arm11)
LIBS=-L$(LIB_USB_PATH) -Wl,-rpath=$(LIB_USB_PATH) –lusb
else
LIBS=-L/usr/lib -lusb
endif
export SUFFIX=so
```

The above makefile statements defined the compilation flag CFLAG, linker flag LFLAGS, and linker argument LIBS. For the i386 architecture, linker linked to the default libusb in the Linux system. For ARM11, linker linked to the compiled libusb.

```
all: libseabreeze.${SUFFIX}
ifeq ($(TARGET),arm11)
libseabreeze.${SUFFIX}: libusb-arm11
else
libseabreeze.${SUFFIX}:
endif
	mkdir -p $(SEABREEZE_LIB)
	+make -C src && (echo; echo Build was successful.; echo)
	$(CPP) $(LFLAGS) -o $(SEABREEZE_LIB)/libseabreeze.$(SUFFIX)
$(SEABREEZE_LIB)/*.o $(LIBS)
.PHONY:libusb-arm11
libusb-arm11:
	$(TCSH) -c "cd $(LIB_USB_DIR) && pwd && ./local-install-arm11"
```

The above makefile statements constructed the default target to build the seabreeze library. If the target architecture was ARM11, the libusb was also compiled.

B. The test makefile

The test/makefile contained a test target and macro definitions for the test program. In this experiment, the default test program was seabreeze_test.

```
OBJS = $(PLATFORM)/seabreeze_test.o \
       $(PLATFORM)/spectral_correction.o
LINKER_REC_PATH = -Wl,-rpath=$(SEABREEZE_LIB) \
                  -Wl,-rpath=\$$ORIGIN/../lib/
APP_PATH = $(shell pwd)/install/$(PLATFORM)/app
LIB_PATH = $(shell pwd)/install/$(PLATFORM)/lib
```

The previous makefile statements defined the application and related library installation paths, a list of objects for the target application, and the definition for the linker record path. The linker record path recorded a relative path from the installation of the application as $ORIGIN/../lib, which told the loader to search for the shared library in the path ../lib. This path was relative to the installation directory for the target application.

```
all: $(PLATFORM)/seabreeze_test
$(PLATFORM)/seabreeze_test : $(OBJS)
      ${CC} $^ -o $@ -L$(SEABREEZE_LIB) $(LINKER_REC_PATH) -lseabreeze -
lusb -lstdc++

$(OBJS): $(PLATFORM)/%.o : %.c $(PLATFORM)
      ${CC} ${CFLAGS} -o $@ $<
$(PLATFORM):
      mkdir -p $(PLATFORM)
```

The previous makefile statements defined the default rule for building the test application. This created a platform directory and built all the objects and executable under that platform directory.

```
install: make_dir cp_seabreeze cp_usb cp_app
      @echo "Installation done."
make_dir:
      /bin/mkdir -p $(APP_PATH)
      /bin/mkdir -p $(LIB_PATH)
cp_seabreeze:
      /bin/cp $(SEABREEZE_LIB)/*.so  $(LIB_PATH)
ifeq ($(TARGET),arm11)
cp_usb:
   /bin/tcsh -f -c "cd $(LIB_USB_PATH) && tar -cvf $(LIB_PATH)/libusb.tar
*"
   /bin/tcsh -f -c "cd $(LIB_PATH) && tar -xvf libusb.tar && /bin/rm
libusb.tar"
else
cp_usb:
   @echo "Nothing to do for libusb installation for $(TARGET)"
endif
cp_app: $(PLATFORM)/seabreeze_test
     /bin/cp $(PLATFORM)/seabreeze_test $(APP_PATH)
```

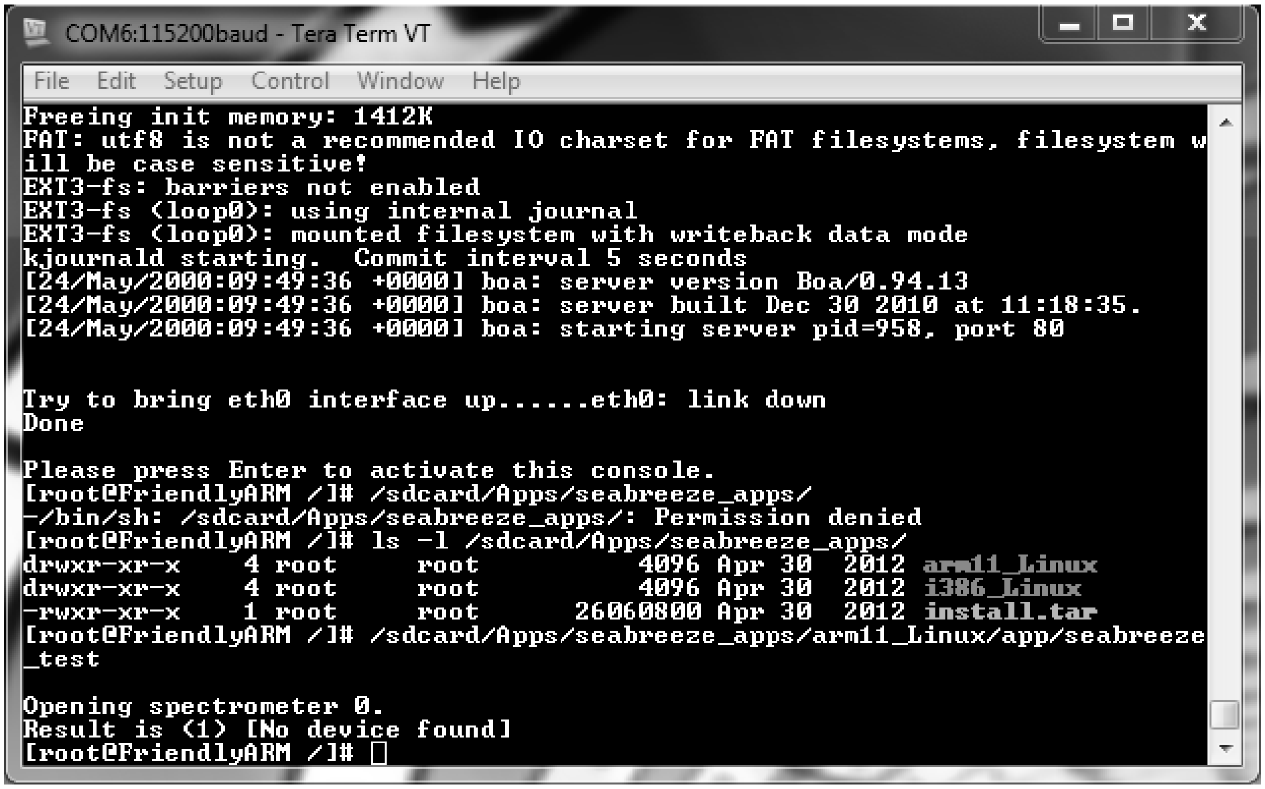
The previous makefile statements defined the installation target actions. It created installation directory structures and copied the required binaries into the structure. For the ARM11 architecture, libusb libraries were also copied along with the seabreeze library.

C. Compilation script for libusb

A new shell script was added into the $SEA_BREEZE_DEV_ROOT/libusb/libusb-1.0.8 directory to compile and install libusb into the ARM11 architecture.

```
#!/usr/bin/tcsh -f
/bin/rm -rf install
/bin/mkdir -p install
./configure --host=arm-Linux --prefix=`pwd`/install
make clean
make
make install
```

The above script removed and created a new directory named “install.” The next step was to configure the build for ARM11 using “--host=arm-Linux-” and local installation with the --prefix=`pwd`/ install option. After the configuration was done, it cleaned, built, and installed libusb. This script made targets that were defined in the Makefile shipped with the original source.


COM6:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
Freeing init memory: 1412K
FAT: utf8 is not a recommended IO charset for FAT filesystems, filesystem w
ill be case sensitive!
EXT3-fs: barriers not enabled
EXT3-fs (loop0): using internal journal
EXT3-fs (loop0): mounted filesystem with writeback data mode
kjournald starting.  Commit interval 5 seconds
[24/May/2000:09:49:36 +0000] boa: server version Boa/0.94.13
[24/May/2000:09:49:36 +0000] boa: server built Dec 30 2010 at 11:18:35.
[24/May/2000:09:49:36 +0000] boa: starting server pid=958, port 80


Try to bring eth0 interface up......eth0: link down
Done

Please press Enter to activate this console.
[root@FriendlyARM /]# /sdcard/Apps/seabreeze_apps/
-/bin/sh: /sdcard/Apps/seabreeze_apps/: Permission denied
[root@FriendlyARM /]# ls -l /sdcard/Apps/seabreeze_apps/
drwxr-xr-x    4 root     root          4096 Apr 30  2012 arm11_Linux
drwxr-xr-x    4 root     root          4096 Apr 30  2012 i386_Linux
-rwxr-xr-x    1 root     root      26060800 Apr 30  2012 install.tar
[root@FriendlyARM /]# /sdcard/Apps/seabreeze_apps/arm11_Linux/app/seabreeze
_test

Opening spectrometer 0.
Result is (1) [No device found]
[root@FriendlyARM /]# 
```

Fig. 42. HyperTerminal for Test App Run

IV. Testing Spectroscopy Application

In this testing, the ARM11-compiled test application was invoked in the Tiny6410 system without the actual spectrometer attached. This was to ensure that compilation and installation were all right for the ARM11 architecture. Fig. 42 is a screen shot of such a testing.

V. Conclusion

With all the experiments done with the spectrometer, it could be concluded that the spectrometer integration with Tiny6410-embedded platform was ready to be used.

APPENDIX C

NFS/SSH Server Setup Using ARM11 Board as Client

I. INTRODUCTION

The Tiny6410, as shown in Fig. 43, is a development board from FriendlyARM 51 using the SMDK6410 platform from Samsung (http://www.samsung.com/us). The processor on this board is an ARM11 implementation from Samsung Inc.

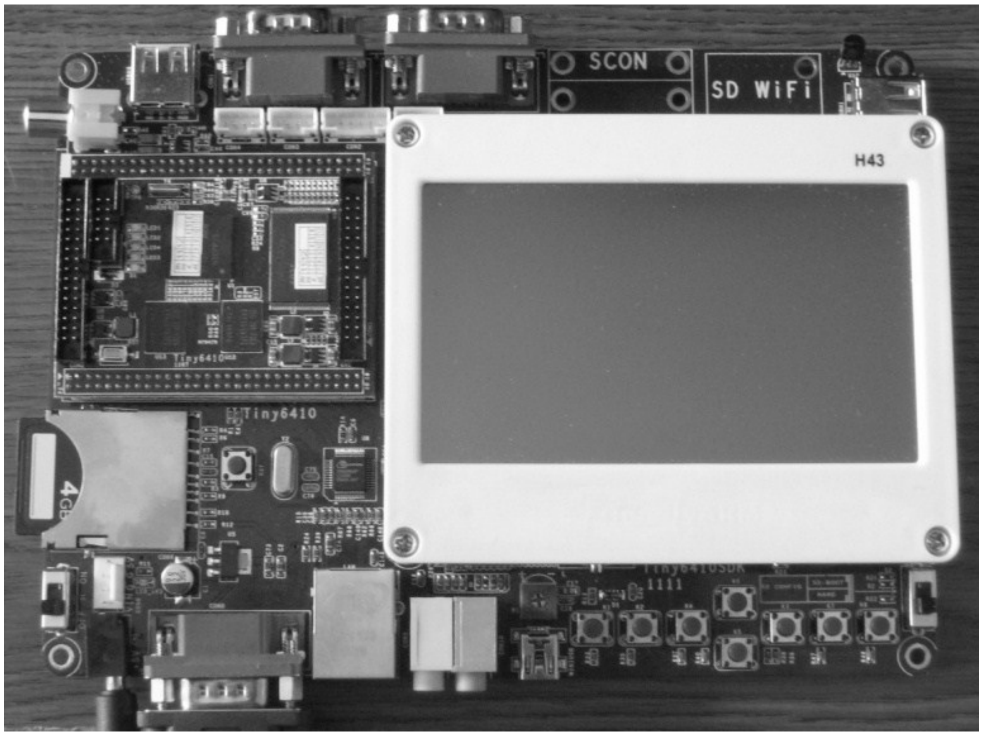


Fig. 43. Tinys6410 SDK

This paper discusses how to set up a network system using Tiny6410 as the client and one server for NFS and the other server for the application server. The first section describes the overall system to be implemented. The second section describes how to set up the server side of the system. The third section discusses the client side changes for Tiny6410 to set up the board as NFS and the application server client.

II. Overall System Design

A. The Physical System

The overall proposed system contained one file server hosting NFS service, one application server, one router, and the Tiny6410 board. Fig. 44 depicts the physical connection between the components.

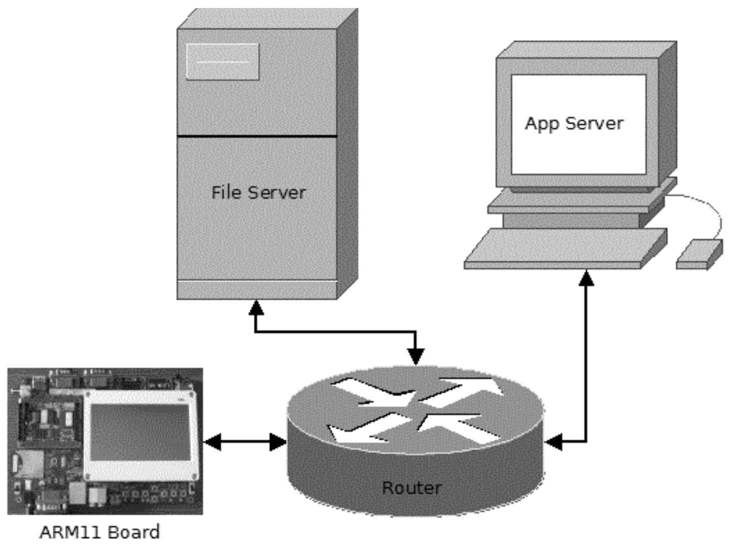


Fig. 44. NFS/SSH-Based System Design

A file server, an application server, and an arm board were physically connected to the router. This way, all the components could communicate with each other.

B. The system concept

At the conception level, three components—the file server, application server, and Tiny6410—can communicate with each other for storage sharing and the remote application execution. Fig. 45 depicts this concept.

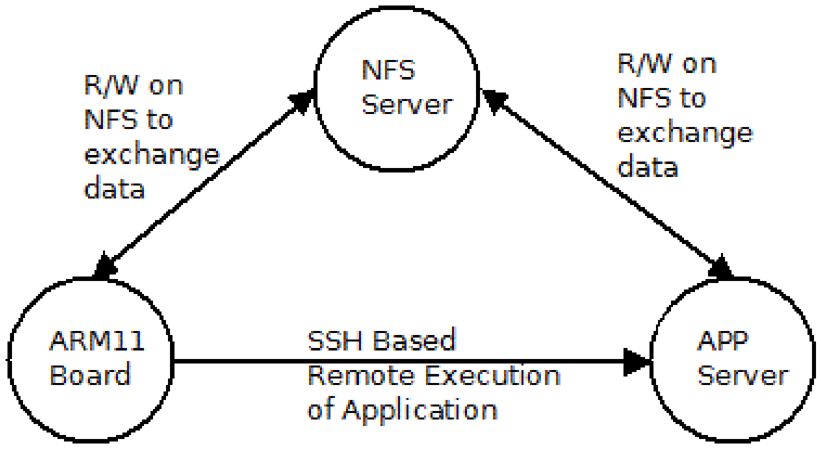

NFS Server
R/W on NFS to exchange data
R/W on NFS to exchange data
ARM11 Board
SSH Based Remote Execution of Application
APP Server

Fig. 45. Overall System Concept

The NFS server hosted the NFS using its location disk. This storage could be accessed by the application server and the Tiny6410 board to read and write files. The application server was configured to handle remote requests to run specific applications using an SSH tunnel. This way the application server was protected from any unauthorized access from within the network. The Tiny6410 board was configured with clients for mounting the NFS and executing remote applications using the SSH tunnel. This allowed the Tiny6410 board to be attached to a network and to access all the services offered by the servers in the network.

III. SERVER SIDE CONFIGURATION

Both the servers described here ran the UBUNTU Linux operating system as the following version.

```
$> uname -a
Linux boson-srvr-01 3.0.0-17-generic #30-Ubuntu SMP Thu Mar 8 17:34:21
UTC 2012 i686 i686 i386 GNU/Linux
```

This experiment used an NFS server named boson-srvr-01 and an application server named boson-srvr-02. The LAN was set up with a dynamic allocation of address (DHCP) for the servers.

A. Configuring the application server

The application server was configured to accept any remote execution request of the application though an SSH tunnel. The configuration steps were taken from the UBUNTU document regarding the OpenSSH server [67].

The first step was to install the OpenSSH server and the client for completeness of the installation.

```
$> sudo apt-get install openssh-client
$> sudo apt-get install openssh-server
```

Once the server package was installed, there was need to modify/add the following into the /etc/ssh/sshd_config file:

```
$> sudo vi /etc/ssh/sshd_config
$> sudo cat /etc/ssh/sshd_config | grep Pubkey
PubkeyAuthentication yes
$> sudo cat /etc/ssh_sshd_config | grep RSAAuth
RSAAuthentication yes
```

The SSH server was required to be restarted after modifying the configuration file. However, this step was not required for every rebooting of the system because the configuration was loaded upfront during the system boot time.

```
$> sudo /etc/init.d/ssh restart
```

B. *Configuring the NFS Server*

The NFS server was configured to share a specific directory on the server local disk over a LAN. This specific directory then could be accessed from NFS clients within the same LAN. The steps to set up the NFS server were taken from UBUNTU documents titled SettingUpNFSHowTo [68].

The first step was to obtain and install the NFS server components:

```
$> sudo apt-get install rpcbind nfs-kernel-server
```

The next step was to make a sharing directory and make it accessible to anyone. However, instead of making this shared directory accessible to everyone, steps were taken to make the sharing more secure, but this was not within the scope of this experiment. For the purposes of this experiment, access permission for everyone was acceptable.

```
$> sudo mkdir -p /export/cod_ann_fs
$> sudo chmod -R 777 /export
```

Once the shared directory was created, the next step was to add an NFS sharing definition to the /etc/exports file. The content looks like the following:

```
$> sudo cat /etc/exports
.  .  .
/export
      192.168.1.0/255.255.255.0(rw,sync,no_subtree_check)
/export/cod_ann_fs
      192.168.1.0/255.255.255.0(rw,sync,no_subtree_check)
.  .  .
```

The above two entries enabled the NFS server to share the directory "/export" and "/export/cod_ann_fs" with any machine having an address starting with 192.168.1. For example, with this configuration, a machine having 192.168.1.149 can access (both read and write) the NFS shared directory. Once the configuration was in place, sharing was enabled with the following command.

```
$> sudo exportfs -ra
```

To test this sharing for the first time, the following services were restarted. However, this was not necessary once the server machine was rebooted.

```
$> sudo /etc/init.d/portmap restart
$> sudo /etc/init.d/nfs-kernel-server restart
```

IV. Client Side Configuration

A. Mounting NFS Partition

The first step was to create a client directory that could be mounted as the report NFS directory. Usually the same name as the NFS directory is given. The directory permission was open for all to read, write, and execute. However, more strict access to the mounted directory may be set up.

```
$> sudo mkdir –p /export/cod_ann_fs
$> sudo chmod –R 777 /export
```

Once the mount partition was identified, there were two ways to perform the NFS mounting. One way was to use explicit options in the command line. The other way was to use /etc/fstab file to put the mount point definitions and mount the NFS with a simpler command line.

The following was the command line with an explicit NFS mount point definition.

```
$> mount -orw,nolock,intr \
         -t nfs \
         boson-srvr-01:/export/cod_ann_fs \
         /export/cod_ann_fs
$> cd /export/cod_ann_fs/
$> ls
authorized_key dropbear-pkg notes z
```

The alternate way to mount NFS was to use a definition entry in the /etc/fstab file. The following was the entry into the file for the same mount point definition, as in the earlier example.

```
$> cat /etc/fstab
# Mount the export dir
boson-srvr-01:/export/cod_ann_fs /export/cod_ann_fs nfs
rw,nolock,intr 0 0
```

The following was the command to mount the NFS with a simpler command.

```
$> mount boson-srvr-01:/export/cod_ann_fs
$> cd /export/cod_ann_fs/
$> ls
authorized_key dropbear-pkg notes z
```

After mounting the NFS, all the standard file system operations could be done on this directory. The following is an example of such operations on the NFS-mounted directory.

```
$> cd /export/cod_ann_fs/
$> pwd
/export/cod_ann_fs
$> du -k .
Filesystem           1K-blocks      Used Available Use% Mounted on
boson-srvr-01:/export/cod_ann_fs
                       7034240   1847712   4833664  28%
/export/cod_ann_fs
$> cat z
```

```
HELLO WORLD!!!

boson-lnx-01

Friendly ARM
```

B. Installing, configuring, and using the SSH client

The Tiny6410 development board ran a variation of the Linux system as follows:

```
$> uname -a

Linux FriendlyARM 2.6.38-FriendlyARM #16 PREEMPT Sun May 27 12:41:24
PDT 2012 armv6l GNU/Linux
```

This Linux installation contained Busybox [70], which is an umbrella application containing most of the commonly used Linux commands. However, it does not contain advanced utilities like an SSH server/client, PERL scripting language, and so on. The Busybox page [70] contains a pointer to an alternate application Dropbear [69], which is equivalent to the SSH server/client yet small enough for ARM11 development board usage.

The standard Dropbear tool contained five utilities as shown in Table 12.

Table 12. Dropbear Utility Names and Usage

Utility name	Purpose
dropbear	The SSH server program to allow Tiny6410 to accept remote execution requests through SSH.
dbclient	The SSH client program to allow Tiny6410 to send remote execution requests to a remote host through SSH.

Utility name	Purpose
dropbearkey	To generate a private-public key pair for auto authentication.
dropbearconvert	To convert the dropbear public key to a standard SSH public key.
scp	A secure copy application program for Tiny6410.

Before the actual installation, an installation directory was created in the NFS partition so the application could be cross compiled on a host machine and shared with the Tiny6410 board through the NFS partition. The following NFS directory was created on the host machine where the dropbear applications were cross compiled.

```
$> mkdir -p /export/cod_ann_fs/arm-Linux/bin
```

The following steps were used to extract the dropbear application source from the downloaded tar file dropbear-2012.55.tar.gz and configure the package for a cross compilation. It was assumed that all the cross compilation tools were available in the default path with a prefix of “arm-Linux-” (e.g., arm-linux-gcc).

```
$> tar -zxvf dropbear-2012.55.tar.gz
$> cd dropbear-2012.55
$> ./configure --host=arm-Linux
```

The next step was to clean and rebuild the dropbear application with cross compilation. In this example, all the dropbear utilities were statically linked to eliminate any library dependency at run time on the Tiny6410 execution environment.

```
$> make clean

$> make  PROGRAMS="dropbear dbclient dropbearkey dropbearconvert scp"
STATIC=1
```

Once the utilities were built, they required stripping to obtain a smaller execution footprint. This was needed to run dropbear utilities in a memory-restrictive environment such as Tiny6410. The following steps were used to strip and install the utilities into the installation location.

```
$> make strip

$> foreach exefile ( dropbear dbclient dropbearkey dropbearconvert
scp )

foreach? cp $exefile /export/cod_ann_fs/arm-Linux/bin

foreach? end
```

At this point, all the installed utilities could be verified as stripped and statically linked as follows:

```
$> file /export/cod_ann_fs/arm-Linux/bin/*

/export/cod_ann_fs/arm-Linux/bin/dbclient:        ELF 32-bit LSB
executable, ARM, version 1 (SYSV), statically linked, for GNU/Linux
2.6.27, stripped

/export/cod_ann_fs/arm-Linux/bin/dropbear:        ELF 32-bit LSB
executable, ARM, version 1 (SYSV), statically linked, for GNU/Linux
2.6.27, stripped

/export/cod_ann_fs/arm-Linux/bin/dropbearconvert: ELF 32-bit LSB
executable, ARM, version 1 (SYSV), statically linked, for GNU/Linux
2.6.27, stripped

/export/cod_ann_fs/arm-Linux/bin/dropbearkey:     ELF 32-bit LSB
executable, ARM, version 1 (SYSV), statically linked, for GNU/Linux
2.6.27, stripped

/export/cod_ann_fs/arm-Linux/bin/scp:             ELF 32-bit LSB
executable, ARM, version 1 (SYSV), statically linked, for GNU/Linux
```

```
2.6.27, not stripped
```

Once installed, dropbear could be used on the Tiny6410 execution environment. The following steps were used to configure Tiny6410 for auto authentication of any SSH-based remote execution. Using these steps, a private-public key pair was generated on the Tiny6410, and the public key was copied to the remote server so that any remote execution from Tiny6410 would be honored without asking for a password. These steps were taken based on an article published on the Internet [71]. In this example, the NFS directory was used to move the public key to the remote server. However, there are more secure ways to accomplish the same task. The client side command was as follows:

```
[FriendlyARM]# dropbearkey -t rsa -f ~/.ssh/id_rsa

[FriendlyARM]# dropbearkey -y -f ~/.ssh/id_rsa | grep "^ssh-rsa" >>
/export/cod_ann_fs/authorized_key
```

The server-side command was as follows:

```
boson-srvr-02> cat /export/cod_ann_fs/authorized_key >>
~/.ssh/authorized_keys
```

Once done, a remote execution request was generated from Tiny6410 as follows:

```
[FriendlyARM]# dbclient -i ~/.ssh/id_rsa -l kpatra boson-srvr-02 cat
/export/cod_ann_fs/z

HELLO WORLD !!!

boson-lnx-01

Friendly ARM
```

The command dbclinet was equivalent to the SSH command. The option "-i ~/.ssh/id_rsa" was used to pass the private key file, unlike the SSH application, which found the private key from the ~/.ssh directory. The option "-l kpatra" requested the server to execute the following command as the user kpatra. The server was required to have the corresponding user account.

V. Conclusion

With the completion of this experiment, two objectives were accomplished. The first was the setup of a Tiny6410-accessible NFS directory. The second was the setup of an SSH-based remote execution from Tiny6410. These two features opened up many possibilities including running complex programs on Tiny6410 through remote execution and storing /sharing large amounts of data without consuming any storage resource onboard.

APPENDIX D

Introduction to the FANN Software Library Package

I. INTRODUCTION

The study of artificial neural networks (ANNs) and their engineering applications has been going on for decades. McCullouch and Pitts [46] proposed the first mathematical model for a neural network in 1943. Later, Rosenblatt [47] developed a perception model to implement a pattern-recognition algorithm based on a two-layer learning computer network. However, neural network research was halted from 1969 for almost 6 years when Minsky and Papert [48] pointed to two key issues in neural network computation machines. The first issue was the inability of a single perception network model to process XOR logic. The second issue was the lack of a powerful-enough computer to process a large neural network. In 1975, Kunihiko Fukushima [49] designed cognition, which was a precursor to a trainable multilayered neural network. Later, rediscovery of a back propagation algorithm for neural network training popularized neural network research and application in engineering problem solving [18].

Many software applications have been developed to implement neural networks. The Fast Artificial Neural Network (FANN) is an open-source software library [60] that implements generalized feed forward ANNs. This library application can implement neural networks of any desired size by selecting an application-specific number of inputs, outputs, hidden layers, and hidden neurons per layer. Several other parameters can be adjusted per the application's needs.

In the following sections, download and installation instruction, API usage, and software directory structure are described. All the instructions documented in this paper are based on the Ubuntu SMP Thu Mar 8 17:34:21 UTC 2012 i686 i686 i386 GNU/Linux operating system. The command examples are based on tcsh as the shell for command-line input.

II. Download and installation

A. Downloading the package

The latest package can be downloaded from the following link:

http://sourceforge.net/projects/fann/files/fann/2.2.0/FANN-2.2.0-Source.zip/download

This downloads the file FANN-2.2.0-Source.zip. The UNIX command "unzip" is used to unzip this file.

```
$> unzip FANN-2.2.0-Source.zip
```

A new directory FANN-2.2.0-Source is created after unzipping the file.

B. Installation

As a pre-requisite, the system must have cmake, make, gcc, and g++.

```
$> which cmake make g++ gcc
/usr/bin/cmake
/usr/bin/make
/usr/bin/g++
/usr/bin/gcc
```

Installation can be done at the system's /usr/local/lib dir, which needs super user privileges (by running the installation command via sudo command in UBUNTU Linux). This is the default mode of installation. There is also another scope to install it in a user-defined library directory. The following shows the installation steps using the default location.

```
$> cd FANN-2.2.0-Source
$> sudo make install
```

This install step creates the artifacts in the system in the following two directories.

• /usr/local/lib	The entire shared library files go here.
• /usr/local/include:	The entire header files go here.

There are four files (which are soft links) inside the /usr/local/lib directory, which are important to know.

1. libfann.so	Same as libfloatfann.so
2. libfloatfann.so	FANN library with floating point.
3. libdoublefann.so	FANN library with double precision floating point.
4. libfixedfann.so	FANN library with fixed point.

To test the installation, the following steps must be performed successfully:

```
$> setenv LD_LIBRARY_PATH /usr/local/lib
$> cd ./examples
$> make runtest
```

To install FANN in an alternate location, the following cmake command must be used instead of the standard cmake command mentioned earlier.

```
cmake -D CMAKE_INSTALL_PREFIX:PATH=/home/abc/lib.
```

III. FANN API USAGE

The FANN library provides two modes of operation:

1. Training mode: In this mode, the application creates, trains, and stores the neural network.
2. Operation mode: In this mode, the application loads a pretrained neural network used to compute output.

In general, for a target application there should be two different programs implemented. The first one is the training program. The purpose of the training program is to create and train the ANN with an existing input and desired output. Once trained, meaning the connection weights are adjusted to produce the desired result for a set of inputs, the ANN is stored in a file to be used in the real application program.

The second type is the target application program. This program typically loads an already stored trained neural network and uses it to compute the result of given inputs for the application. Fig. 46 shows the overall steps for the two modes. The application program has to have the fann.h header file to include the prototype definition of any

FANN API needed. In addition, it needs to link to the shared library of FANN to access the APIs at run time.

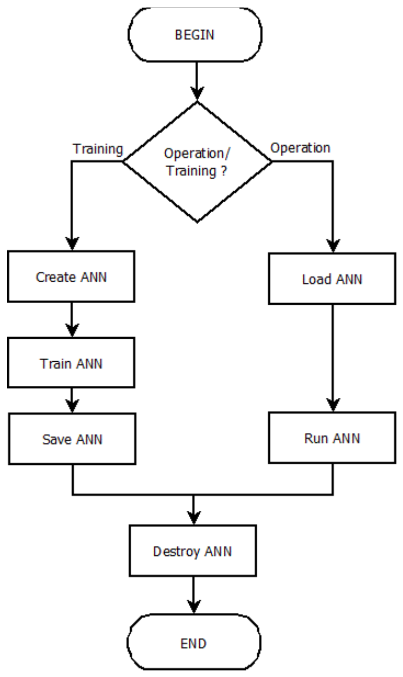


Fig. 46. FANN Usage Steps.

A. Training mode APIs

Table 13 shows the four available APIs needed to perform the steps in training mode.

Table 13. API List of ANN Training

API	Description
fann_create	Creates and returns the pointer to struct fann. The parameters to be passed into this API are connection rate (1.0 for fully connected network), learning rate, number of layers, and number of neurons per layer starting with the input layer and ending with the output layer.
fann_train_on_file	Trains a given network with a given training data file. The parameters to be passed are the network, train file, max number of epochs, number of epochs to produce a status, desired mean square

API	Description
	error. The training steps at max epoch or mean square error are lower than the desired value, which ever happens first.
fann_save	Saves the network in a text file. The parameters passed are the network and file name to the text file where the network is to be stored.
fann_destroy	Cleans the memory allocation during the previous steps. This should happen just before quitting the application.

The following are examples and explanations of sample usage of each of the above APIs.

```
1. struct fann *ann = fan_create(1.0, 0.7, 3, 2, 2, 1);
```

The previous function call will create an ANN with full connection (first argument), learning rate 0.7 (second argument), 3 layers (third argument), 2 neurons in input layers (fourth argument), 2 hidden neurons in one hidden layer (fifth argument), and one neuron at output layer (sixth argument).

```
2. fann_train_on_file(ann, "xor.data", 500000, 1000, 0.0001);
```

The previous function call will train ANN with the data from the xor.data file with maximum epoch (during one epoch, each of the training pairs are trained for one iteration) of 50,000 (third argument), reporting training status between epochs as 1,000 (fourth argument) and with a desired mean square error as 0.0001 (fifth argument). With this specific example, the training will stop when either the number of epochs is greater than 50,000 or the desired mean square error is below 0.0001, whichever happens first.

```
3. fann_save(ann, "xor_float.net");
```

The previous function saves the trained ANN as xor_flaot.net inside a text file.

```
4. fann_destroy(ann);
```

The previous function destroys all the memory allocated by the ANN structure. This is usually the last call before returning from the main routine of the application.

B. Operation mode APIs

The Table 14 lists the three APIs needed to perform the steps in training mode.

Table 14. FANN API List of ANN Operation

API	Description
fann_create_from_file	Creates and returns pointer to struct fann, reading the network definition from file. The parameter to be passed into this API is the ANN definition file name.
fann_run	Computes and returns the output for a given input.
fann_destroy	Cleans the memory allocation during the previous steps. This should happen just before quitting the application.

An example of the two new APIs is discussed in this subsection. Usage example for fann_destroy was shown in the previous subsection.

```
1. struct fann *ann = fann_create_from_file("xor.net");
```

The previous function call creates a FANN structure by reading the network definition from the xor.net files. The format of the definition is not important in this context. The recommendation is to create such a definition only by using the fann_save API during the training mode operation of the application.

```
fann_type input[2];
input[0] = 1;
input[1] = -1;
fann_type *calc_out = fann_run(ann, input);
```

The previous piece of code computes the result and returns it as an array into calc_out (which is at least an array containing one element). The arguments are ANN and the input values for the computation.

C. Advance Usage of API

The FANN library supports adjustment of the following ANN parameters.

1. Activation method for both hidden and output neuron layers.
2. Steepness parameter used in the sigmoid family activation function.
3. Learning rate of the neural network.
4. Initial weight of the connection. By default, it is set randomly between –1.0 and 1.0.

During the network-training phase, it is sometimes necessary to adjust these advanced parameters to fine-tune the outcome of the network. Table 15 shows the APIs to adjust such advance parameters.

Table 15. FANN API List of ANN Operations

API	Description
fann_set_learning_rate	Sets the learning rate of an ANN. The arguments are the network and the learning rate. fann_set_learning_rate(ann,0.55)
fann_randomize_weight	Sets the user-defined random weight boundary for the connections. The arguments are the network, lower boundary, and upper boundary. fann_randomize_weight(ann,-0.6,0.8)
fann_set_activation_function_hidden	Sets the activation function for the hidden layers. The arguments are the network and the function type. fann_set_activation_function_hidden (ann,FANN_SIGMOID_STEPWISE)
fann_set_activation_function_output	Sets the activation function for the output layers. The arguments are the network and the function type. fann_set_activation_function_output(ann, FANN_SIGMOID_SYMMETRIC)
fann_set_activation_hidden_steepness	Sets the activation steepness for the sigmoid family activation functions for the hidden layer. fann_set_activation_hidden_steepness(ann,0.5)
fann_set_activation_output_steepness	Sets the activation steepness for the sigmoid family activation functions for the output layer. fann_set_activation_output_steepness(ann,0.8)

The following are the different activation functions supported by the FANN 2.2.0 library package.

```
FANN_LINEAR                               FANN_GAUSSIAN_STEPWISE,
FANN_THRESHOLD                            FANN_ELLIOT
FANN_THRESHOLD_SYMMETRIC                  FANN_ELLIOT_SYMMETRIC
```

```
FANN_SIGMOID                              FANN_LINEAR_PIECE
FANN_SIGMOID_STEPWISE                     FANN_LINEAR_PIECE_SYMMETRIC
FANN_SIGMOID_SYMMETRIC                    FANN_SIN_SYMMETRIC
FANN_SIGMOID_SYMMETRIC_STEPWISE           FANN_COS_SYMMETRIC
FANN_GAUSSIAN                             FANN_SIN
FANN_GAUSSIAN_SYMMETRIC                   FANN_COS
```

D. FANN Train File Format

Because the training files are to be created by the application users, it is important to describe the training file format in the context of FANN API usage. The FANN training file has a very simple format but is not very flexible (there is no support for the comment or blank lines). The first line has three numbers that are space- or tab-separated (a data set composed of the input pattern and output pattern), the number of inputs, and the number of outputs. From the next line onward, each of the two lines composes one data set. In a data set, the first line has the space- or tab-separated input values and the second line has the space- or tab-separated output values. Table 16 shows an example of the training set for XOR training.

Table 16. Training Set for XOR Training

Content	Explanation	Section
4 2 1	4 data set with 2 inputs and 1 output	Definition line
1 –1	Input 1 and –1	Data Set 1
–1	Output –1	
–1 1	Input –1 and 1	Data Set 2
1	Output 1	
1 –1	Input 1 and –1	Data Set 3
1	Output 1	

Content	Explanation	Section
1 1	Input 1 and 1	Data Set 4
–1	Output –1	

IV. FANN Package Directory structure

It is very important to know the directory structure of the unzipped package in order to change or experiment with the source code and examples. Table 17 shows the subdirectory and its content upon extracting the zip file.

Table 17. FANN 2.2.0 Directory Structure

Dir name / file name	Purpose
bin	Directory storing precompiled libraries and executable.
CMakeLists.txt	Configuration file for cmake.
COPYING.txt	Copyright statement file.
examples	Directory-storing example application programs using FANN library.
src	Directory storing source code and header files (inside the subdirectory named include) for FANN library.
cmake	Directory-storing files required for cmake.
VS2010	Directory-storing Visual Studio 2010 project definitions.
datasets	Directory-storing training and testing file for the examples.
README.txt	The README file.

V. Conclusion

With all the experiments done with the FANN library, it can be concluded that the FANN library can be successfully used to create and use an ANN.

APPENDIX E

Extending FANN Library for Compact C-model Generation

I. INTRODUCTION

The fast artificial neural network (FANN) library [60] provides the functionality to create, train, store, and reload a neural network for application-implementing solutions using a neural network technique. It stores the network definition in a text file. Using the provided API, the trained network can be reloaded for the purposes of application usage. However, reusing the neural network through the native API is costly in the sense that the application needs to

1. Load all the APIs in FANN unnecessarily; most of them are not needed.
2. Create a lot of pointer-based structures to represent a neural network, which is not necessary while using the neural network for computation purposes.

Often in the case of embedded applications, it is good to keep the memory footprint to a minimum and within a predictable, executable size. This document describes an extension to the FANN 2.2.0 library to generate a compact C-model, which can be integrated into the application at the source-code level. This compact C-model does not need any of the APIs from the original library and it represents the neural network purely with static arrays. Hence, the application memory footprint is completely predictable. It is the size of the target-compiled application (the assumption is that a target application only uses static arrays).

II. Basic Architecture For the ANN C-Model

A. General ANN architecture

Fig. 47 depicts a neural network model with two inputs, one output, and one hidden layer containing two neurons. An ANN contains at least one input layer and one output layer. It may contain one or more hidden layers. Each layer contains multiple neurons and one bias neuron (marked as *b* in the diagram). Each neuron presents a numerical value for the next layer.

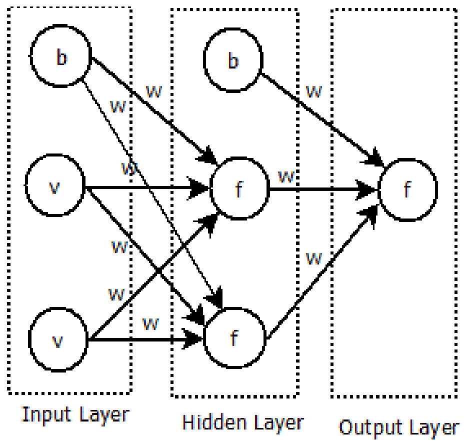


Fig. 47. Basic ANN Architecture

For a fully connected network, each neuron in a noninput layer is connected to all the neurons in the previous layer with different weights. The input to a neuron from its predecessor neuron is the product of the value from its predecessor neuron and the connection weight *w*. The neurons in the input layer and bias neurons usually do not contain any activation function or linear function. The bias neuron's value is usually set to a singular value of 1.

B. The equivalent C-model architecture

The fundamental motivation to design a C-model for ANN is to represent the target ANN structure with static arrays. Using that approach, ANN size can be very compact and predictable, which is a suitable model for embedded application usage. From the general ANN architecture, it can be concluded that there are three distinct components to be represented in a corresponding ANN C-model: the neuron, perception layers, and neuron connections.

Because the goal is to represent an already trained ANN, all the parameters (the activation function, the connection weight, and so on) are known in advance. Hence, it is possible to represent the entire ANN using a group of static arrays. All three distinct components, as described earlier, can be represented with three static arrays. Fig. 48 depicts the basic architectural concept for the ANN C-model, where the dotted connection shows the conceptual relations between different arrays.

In this architecture, all the neurons are placed in one array and are sorted according to their layer position (i.e., neurons in the first layer go first, neurons in the second layer go next, and so on). In addition, the bias neuron for a layer has the highest index in the neuron array compared to other neurons in the same layer. A neuron is a collection of values and activation functions in the C-model. In the example of Fig. 48, there are nine neurons (including the bias neurons). In Fig. 48, all the nine neurons are placed in an array of size nine.

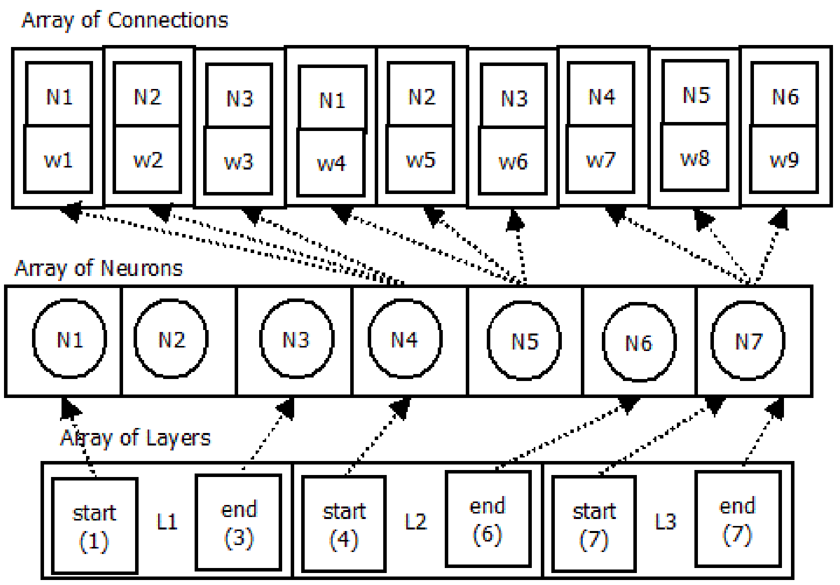


Fig. 48. Top level ANN C-model Architecture

All the layer definitions are placed in another array. The layer definition contains the start neuron index and the end neuron index. The end neuron index always points to a bias neuron in this proposed ANN C-model architecture. In the example, three layers are represented in the array of layers, as shown in Fig. 48.

All the connections are placed in another array where a connection definition contains the index of the source neuron and the connection weight. All the connections are arranged inside the connection array in such a way that the connection sequence corresponds to the neuron sequence as the destination neuron in the connection. The destination neuron sequence starts from the second layer, skipping all the bias neurons. The number of connections from a neuron can be determined from the layer information of the immediately preceding layer. In Fig. 48, there are nine connections, which are represented in the connection array in Fig. 48. According to this architecture, Neuron N4 is connected to N1, N2, and N3 neurons with connection weight w1, w2, and w3 respectively. Inherently, this architecture assumes the full connection rate. However, it is also possible to implement partial connections using a connection weight of 0. Inherently this architecture assumes the full connection rate. However, it is also possible to implement partial connections using a connection weight of 0.

III. THE COMPLETE C-MODEL ECHO SYSTEM

The C-model, described in Section II, is accompanied using a different set of APIs, which together compose the complete ANN C-model echo system. The API system is classified into three module categories: core APIs, IO APIs, and statistics APIs. Each of these categories can be complied and linked as needed by the target application. The core API is independent and can be linked with an application without any other APIs. On the other hand, both the IO API and statistic API need the application to have the core API linked as well. However, the IO and statistic API are independent of each other.

The core API contains the ANN C-model, as described in Section II, and the API to compute the result using ANN C-model. The target application can pass the input to compute the API and get the output back from the module. The output is computed using the ANN C-model. Additionally, this module has other query APIs to get the number of inputs, outputs, layers, and other information about the ANN.

The utility API module contains an API to read the ANN training data file [2] and provide input and expected output to the target application. This API module can be used to test ANN with an existing data set to validate the result.

The statistic API module contains an API to collect all the outputs and maintain an error database. The application can return error statistics (mean relative error and standard deviation of means relative error) and relative error distribution data. The relative error is defined in terms of the percentage of expected output. The relative error ("err") is defined by Equation 14, where *O* is the output and *E* is the expected output.

$$err = \frac{(O - E)}{E} * 100 \qquad \text{Equation 14}$$

If the expected output is 0, both the current output and expected output are added with a unity value so that Equation 1 does not run into an invalid condition.

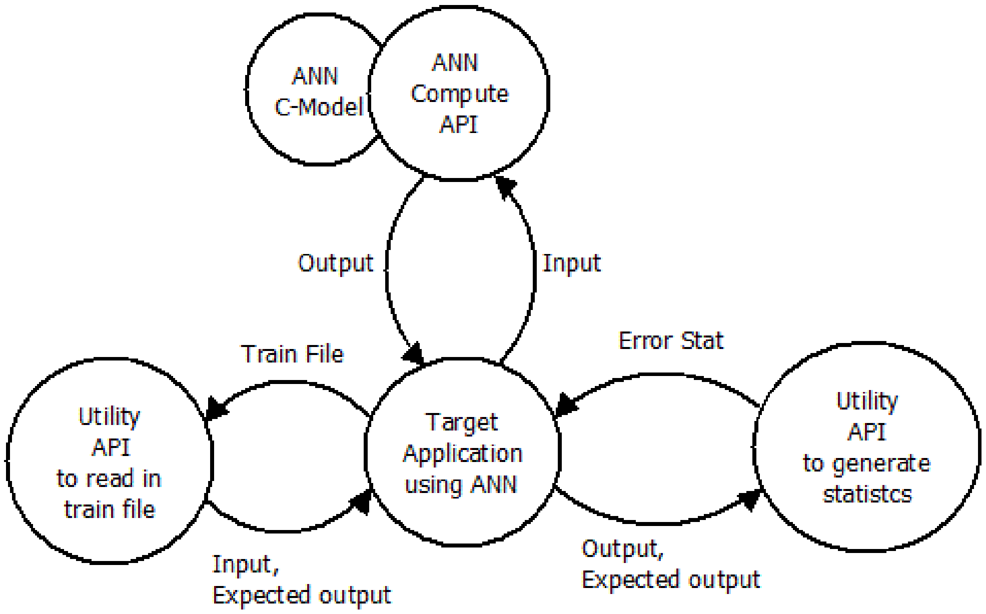


Fig. 49. ANN C-model Echo System

Fig. 49 depicts the complete C-model echo system. The circles represent different modules of the system. The arrows represent the I/O of the module. The ANN C-model, as described in section II, is associated with the ANN compute API module. In section V, the details of the data structure and APIs are discussed.

IV. FANN-2.2.0 EXTENSION

To translate the FANN 2.2.0 internal model into C-model, the following new API has been created, as shown in Table 18.

Table 18. API for ANN C-model Generation

API	Description
fann_save_C	Saves the FANN internal ANN model to a C-model, which can be compiled with any general C compiler, such as gcc. The argument for this API is struct fann *. e.g. fann_save_C(ann)

A new source file (fann_io_embedded.c) and a new header file (fann_io_embedded.h) are created under the FANN-2.2.0 source distribution. In this experiment, instead of a make-based system, Netbeans was used to define the FANN extension project and was compiled using Netbeans GUI. However, because the FANN-2.2.0 source distribution is very simple, it is possible also to extend the existing make/cmake system to include this newly introduced piece of code.

There is another piece of new code: fann_utils.c (and corresponding fann_utils.h) was added to FANN-2.2.0 to implement some common statistic APIs used in both the FANN library and ANN C-model APIs. Details of the APIs are discussed in section V. The only relevant information is that to compile this file as a part of the FANN library, the compiler macro FANN_LIB must be defined (in gcc it is passed using DFANN_LIB).

```
$> gcc –c fann_utils.c -DFANN_LIB –o fann_utils.o
```

The fann_save_C API creates the following list files in the current run dir as shown in Table 19.

Table 19. Files Generated with FANN C-model Generation

File	Description
fann_app_global.h	The C-model definition for ANN.
fann_app.h	The header files of the core API module to be included in the source code of the target application.
fann_app_int.h	This is the internal header file for the core API module.
fann_utils.h	This is the header file of the IO and statistic API module to be included in the target application if needed.
fann_app.c	The implementation for the core API module.
fann_example_app.c	An example application program that can be used as the starting point of an application program using the ANN C-model.
fann_utils.c	The implementation for the IO and statistic API module.
Makefile	A simple GNU makefile to build the target application.

Except for fann_app_global.h, all files are copied from a predefined directory. For this purpose, a new directory "config" is added to the FANN-2.2.0 package directory and compiler macro FANN_COFIG_DIR is used to define this directory location inside the source code. In the Netbeans project definition, the FANN_CONFIG_DIR compiler macro is added so that compilation command has the following switch:

```
-DFANN_CONFIG_DIR="\"/home/kpatra/ThesisProjects/COD-ANN/FANN/FANN-2.2.0-
Source/config\""
```

V. ANN C-MODEL APIS

A. C data structure definition

In the file fann_app_int.h, the data structure for the neuron, layer, and connection has been defined. A neuron data structure is defined as follows:

```
typedef struct {
     int            num_of_input;
     Fann_Actfn_Type     actfn;
     float         activation_steepness;
     float              value;
} Fann_Neurone;
```

A neuron data structure contains information about the number of inputs (num_of_input), activation function (actfn), activation steepness for the sigmoid group of activation functions (activation_steepness), and the current value of the neuron (value).

A layer data structure is defined as follows:

```
typedef struct {
      int   start_neuron_index;
      int     end_neuron_index;
      int     conn_weight_start_index;
} Fann_Layer;
```

A layer data structure contains a star and end neuron index for the layer in the neuron array index (start_neuron_index and end_neuron_index). There is also a start index for the connection in the connection array (conn_weight_start_index). Although this connection array can be determined at execution time, it has been included in the data structure to save some computation time while keeping the memory footprint under control. It is not usual to have a very large number of layers in any ANN architecture; thus this extra storage in the layer data structure should not consume a significant amount of memory.

The connection data structure is defined as following. It contains the source neuron index for the connection and the connection weight. As discussed in section II, ANN data structure indirectly indicates the destination neuron for the connection.

```
typedef struct {
  int       from_neuron_index;
  float     weight;
} Fann_Conn_Info;
```

In the file fann_app_global.h file, the actual C-model for ANN is defined. The following is an example of such a model for XOR implementation using ANN.

```
Fann_Neuronefann_neurones[] = {
    {0, FANN_LINEAR, 0.00000e+00, 0},
    {0, FANN_LINEAR, 0.000000e+00, 0},
    {0, FANN_LINEAR, 0.00000e+00, 1},
    {3, FANN_SIGMOID_SYMMETRIC, 1.00000e+00, 0},
    {3, FANN_SIGMOID_SYMMETRIC, 1.00000e+00, 0},
    {3, FANN_SIGMOID_SYMMETRIC, 1.00000e+00, 0},
    {0, FANN_SIGMOID_SYMMETRIC, 1.00000e+00, 1},
    {4, FANN_SIGMOID_SYMMETRIC, 1.00000e+00, 0},
    {0, FANN_SIGMOID_SYMMETRIC, 1.00000e+00, 1}
};
Fann_Layer         fann_layers[] = {
      { 0, 1, -1 },
      { 3, 5, 0 },
      { 7, 7, 9 }
};
Fann_Conn_Info    fann_connections [] = {
      {0, 1.5397429466247558593 8e+00},
      {1, -2.4302451610565185546 9e+00},
      {2, -1.6280171871185302734 4e+00},
      {0, 9.1879034042358398437 5e-01},
      {1, 1.1830010414123535156 2e+00},
      {2, 2.0600714683532714843 8e+00},
      {0, -1.8916923999786376953 1e+00},
      {1, 1.3889137506484985351 6e+00},
      {2, -1.6691235303878784179 7e+00},
      {3, 3.1712090969085693359 4e+00},
      {4, 2.5778272151947021484 4e+00},
      {5, 3.4204952716827392578 1e+00},
      {6, 8.4149968624114990234 4e-01}
     };
```

B. ANN C-Model APIs

The following are the core APIs that an application should need to use the C-model as shown in Table 20.

Table 20. C-model Core APIs

File	Description
fann_get_number_of_layers	Returns the number of layers.
fann_get_number_of_input	Returns the number of inputs.
fann_get_number_of_output	Returns the number of outputs.
fann_get_number_of_neurons	Returns the total number of neurons.
fann_get_number_of_conns	Returns the total number of connections.
fann_execute	Computes the output for a given input using the ANN. It takes arguments as an array of ouputs and an array of inputs. The output array is populated with computed results after the execution is done. It returns the execution status as Boolean (an enum with values true

File	Description
	and false). e.g., Boolean stat = fann_execute(out,in); // out => array of float, in => array of float

The following are the IO utility APIs included in the application using fann_utils.h, as shown in Table 21.

Table 21. C-model IO Utility APIs

File	Description
fann_utils_open_train_file	To open a FANN training file for reading; the argument is the training file name. The function returns Boolean status upon completion. e.g., stat = fann_utils_open_train_file (“xor.dat”)

fann_utils_get_next_data	To get the next data set (input and expected output) from the opened training file; the arguments are the pointer to the input array and the pointer to the output array. This function allocates the required array inside the function itself so that the application does not need to allocate the array beforehand. This function returns a Boolean status upon completion. If the training file reaches the end, this function will return false and close the file descriptor for the training file.
	e.g., while (fann_utils_get_next_data (in_pattern, out_pattern)) {
	// do something with the data
	// where in_pattern is float **
	// where out_pattern is float **
	}

fann_utils_init_util	THIS FUNCTION IS FOR THE FANN LIBRARY ONLY AND IS ONLY FOR USING THE STATISTICAL UTILITY FUNCTION. This function initializes the internal structures of the utility module for providing the statistical APIs. The arguments are the number of data sets and the number of outputs. It returns a Boolean status upon

File	Description
	completion. e.g., stat = fann_utils_init_util(455, 3);

The following are the statistic utility APIs, as shown in Table 22.

Table 22. C-model Statistics Utility APIs

API	Description
fann_utils_insert_compare_data	To open insert comparison data in statistical database of the utility; the argument is the constant array to expected output and actual output. The function returns a Boolean status upon completion. e.g., stat = fann_utils_insert_compare_data(exp,out);
fann_utils_get_mean_relative_error	Returns mean relative error so far; no argument is needed. e.g. mre = fann_utils_get_mean_relative_error();
fann_utils_get_std_dev_of_relative_err or	Returns standard deviation for relative error; no argument is needed. e.g. mestd= fann_utils_get_std_dev_of_relative_error();
fann_utils_dump_err_stat	Creates an error statistics file named after the given argument. e.g., fann_utils_dump_err_stat("err.stat");

The error statistics file contains tab-separated information about the mean relative error, standard deviation of relative error, and distribution information on the deviation between –100% to +100%. This file can easily be imported into any spreadsheet application for further analysis.

VI. Usage of Generated C-model

A. Building the application program

The generated C-model for ANN also contains a sample application program named fann_example_app.c, which can be used as the starting point for the target application program. This example program can be built with the created Makefile. The Makefile has a target "help" as follows:

```
$> make help

make APP_NAME=<app name> [TRAIN_DATA=<train data file>]
[USE_FANN_UTILS=<YES|NO>] [ STAT_FILE=<stat file> ] [CC_EXT <compiler
prefix>]
```

Table 23 explains each make macro and its usage.

Table 23. Make File Macros for Building an Application Using the ANN C-model

MACRO	Description
APP_NAME	To supply the application name <app_name>; it is assumed that the corresponding source file <app_name>.c exists. This macro has to be used in the Make command line.
TRAIN_DATA	To supply the FANN training data file; this is optional. The default train data file name is SAMPLE.DAT.
USE_FANN_UTILS	An optional macro that takes a YES or NO value. To include the FANN utility APIs (both IO and statistics), this macro must be passed with a YES value. The default value is NO.
STAT_FILE	This is an optional macro to define the generated statistic file name. Only if this macro is defined, the statistics utility code is compiled. The default value is CMODEL.STAT.
CC_EXT	This is an optional macro to define the prefix of the cc

MACRO	Description
	command. This is useful for compiling the application with cross compilers.

The following is an example for compiling an application program generated through FANN extension testing.

```
$> make APP_NAME=fann_example_app \
        TRAIN_DATA="\"xor.dat\"" \
        USE_FANN_UTILS=YES \
        STAT_FILE="\"xor.cmodel.stat\""\
        CC_EXT=arm-Linux-
```

The above example compiles the application with a main source at fann_example_app.c with the arm-Linux-cc compiler, using xor.dat as a training file and xor.cmodel.stat as a statistics file. The application will also include the IO utility and statistics utility module in the compilation.

B. Understanding the example application program

The following is the source code listing for the created example application program.

```
#include <stdio.h>
#include <stdlib.h>
#include "fann_app.h"
#include "fann_utils.h"
#ifndef FANN_TRAIN_DATA
#define FANN_TRAIN_DATA "SAMPLE.DAT"
#endif
```

```
#ifndef FANN_STAT_FILE
#define FANN_STAT_FILE "CMODEL.STAT"
#endif

int main(int argc, char** argv) {
    float *input = NULL;
    float *expected_output = NULL;
    float *output =
malloc(sizeof(float)*fann_get_number_of_output());
    printf("INFO : Number of layers
%d\n",fann_get_number_of_layers());
    printf("INFO : Number of input
%d\n",fann_get_number_of_input());
    printf("INFO : Number of output
%d\n",fann_get_number_of_output());
    printf("INFO : Number of neurons
%d\n",fann_get_number_of_neurones());
    printf("INFO : Number of connections
%d\n",fann_get_number_of_conns());
    printf("\n");
    if (!fann_utils_open_train_file(FANN_TRAIN_DATA)) {
        return EXIT_FAILURE;
    }
    while(fann_utils_get_next_data(&input,&expected_output)) {
        fann_execute(output, input);
        #ifndef NO_FANN_UTIL_STAT
        fann_utils_insert_compare_data(expected_output,output);
        #endif
    }
    #ifndef NO_FANN_UTIL_STAT

    printf("INFO: Dumping error statistics at
    %s\n",FANN_STAT_FILE);
40      fann_utils_dump_err_stat(FANN_STAT_FILE);
41      #endif
43      free(output);
44      return EXIT_SUCCESS;
45  }
```

Table 24 explains each source code line of the example application program.

Table 24. Source Code Explanation for Application Using the ANN C-model

Lines	Description
4-5	Includes core and IO APIs.
8,12	Sets default train data and stat file name.
19	Allocates floating type array with a size equal to the number of outputs of the ANN.
21-27	Prints ANN information.
29	Opens training data file to be read.
32-37	A while loop that continues until all the data are read. It stores the input pattern and expected output pattern.
33	Executes the ANN to compute output into the output array.
35	Inserts the comparison data into a statistical database.
40	Creates the error statistic file.

VII. VALIDITY OF GENERATED C-MODEL

A. General validation strategy

The general strategy to validate the C-model result is to compare it to the output from the sample application using the FANN 2.2.0 library.

It is easier to inspect the output from smaller test cases by reviewing the results. This is not the case for larger test cases. For larger test cases, statistical validation has been adapted to verify and validate the C-model. The mean relative error and standard deviation from the mean relative error are compared between applications using the C-model and the same application using the FANN-2.2.0 library. There should be no difference or very little difference (within 1% of error tolerance) between these statistical metrics. In addition, the error distribution between two runs should be identical or have very small differences.

B. List of test cases

The following are the test cases used to validate the C-model as shown in Table 25.

Table 25. FANN C-model Test Cases

Name	# of inputs	# of outputs	# of layers	# of hidden neurons	# of test data
XOR	2	1	3	3	4
1pClass	2	1	2	0	6
Robot	48	3	3	96	594
Mushroom	125	2	3	32	4062

C. Output Comparison

The following is the statistical comparison between test cases as shown in Table 26.

Table 26. FANN C-model Test Results

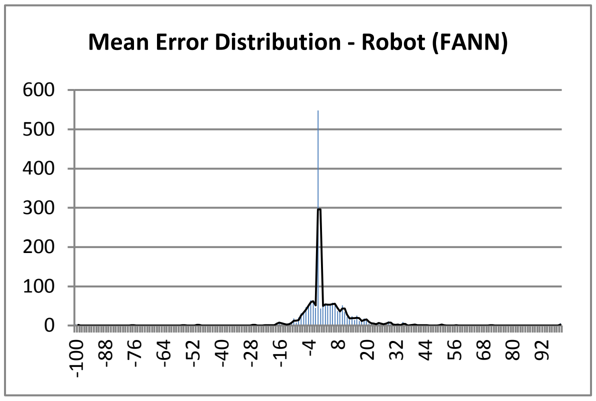
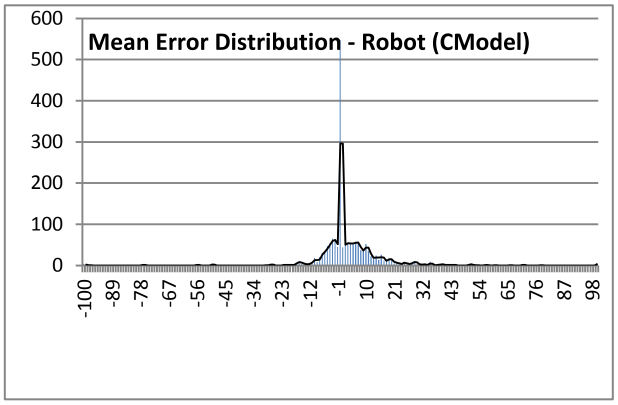
Name	Mean relative error			Standard deviation of relative error		
	FANN	*C-MODEL*	*DIFF*	*FANN*	*C-MODEL*	*DIFF*

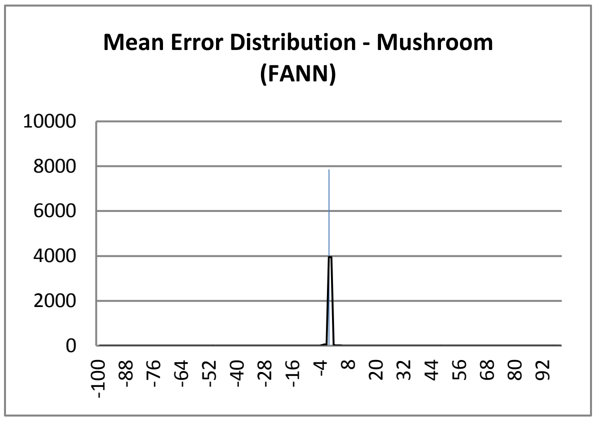
Name	Mean relative error			Standard deviation of relative error		
	FANN	*C-MODEL*	*DIFF*	*FANN*	*C-MODEL*	*DIFF*
XOR	–1.45	–1.45	0	0.31	0.31	0
1pClass	0	0	0	0.6	0.6	0
Robot	3.45	3.45	0	15.2	15.2	0
Mushroom	0.06	0.06	0	2.46	2.46	0

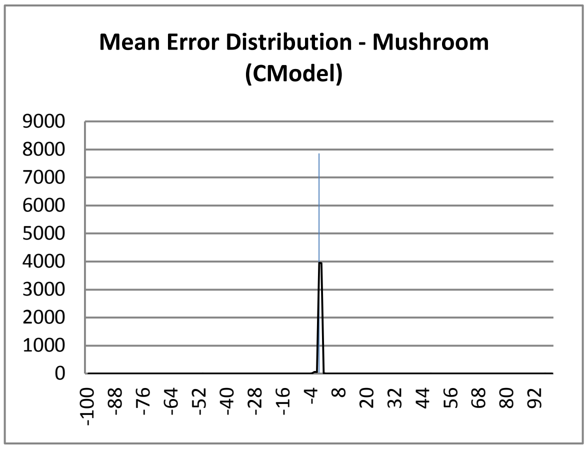
For all the statistical metrics, the results of FANN 2.2.0 and the C-model are identical.

Table 27 shows the mean error distribution between FANN and the C-model for the robot and mushroom test cases.

Table 27. FANN C-model Mean Error Distribution





The following is the output from the XOR C-model application.

```
INPUT: (-1.000000,-1.000000)  OUTPUT: (-0.987475)      EXPECTED: (-
1.000000)   DIFF: (0.012525)

INPUT: (-1.000000,1.000000)   OUTPUT: (0.983460) EXPECTED: (1.000000)
      DIFF: (-0.016540)

INPUT: (1.000000,-1.000000)   OUTPUT: (0.981609) EXPECTED: (1.000000)
      DIFF: (-0.018391)

INPUT: (1.000000,1.000000)    OUTPUT: (-0.989289)      EXPECTED: (-
1.000000)   DIFF: (0.010711)
```

The following is the output from the 1pClass C-model application.

```
INPUT: (1.000000,2.000000)    OUTPUT: (0.998061) EXPECTED: (1.000000)
      DIFF: (-0.001939)

INPUT: (1.000000,3.000000)    OUTPUT: (0.999996) EXPECTED: (1.000000)
      DIFF: (-0.000004)

INPUT: (2.000000,3.000000)    OUTPUT: (0.998061) EXPECTED: (1.000000)
      DIFF: (-0.001939)

INPUT: (2.000000,1.000000)    OUTPUT: (0.001993) EXPECTED: (0.000000)
      DIFF: (0.001993)

INPUT: (3.000000,1.000000)    OUTPUT: (0.000004) EXPECTED: (0.000000)
      DIFF: (0.000004)

INPUT: (3.000000,2.000000)    OUTPUT: (0.001993) EXPECTED: (0.000000)
      DIFF: (0.001993)
```

VIII. Conclusion

The experimental results show that the C-model ANN evaluation is functionally equivalent to the FANN 2.2.0 ANN evaluation.

APPENDIX F

"anCod" A Software for Analyzing Water COD

I. INTRODUCTION

Chemical oxygen demand (COD) is one of the unique parameters for determining water quality. There have been many types of chemical, electrochemical, photochemical, and pure photometry-based procedures to determine the COD content of water. Among these different procedures, photometry-based analysis is the easiest and quickest process, simply because it does not involve any chemical process. This photometry-based analysis exploits the unique spectral signature for water contaminants.

In this project, a software suit has been developed, which can be executed on a cloud-enabled embedded platform using ANN-based COD measurement to collect spectrometry data from a portable spectrometer. The following sections describe the requirement (section II), architecture (section III), user interface (section IV), implementation (section V), and experiment result on training (section VI).

II. REQUIREMENT FOR THE SOFTWARE

The proposed software has three distinct components. One component is responsible for collecting photometry data, sending it over the cloud to a server for data analysis, and collecting the result from the server side. The other component is the server component, which collects the data from the client, runs analysis, and sends the result back. The third component is the ANN model generator using training data. Either the training data are mathematically generated or field collected real with a known water sample COD.

A. Client Component Requirements

The following is the itemized requirement for the client component.

1. The client can be executed on an ARM11-based mobile platform.
2. The client should be able to collect data from the photometry device.
3. The client should be able to store the data on a remote server using the cloud.
4. The client should be able to request the server to start analyzing the collected data.
5. The client should be able to collect the COD results from the server and display the results locally.
6. The client should be able to create ANN training and test data using a mathematical model for the photometry data.
7. The client should be able use collected photometry data for analyzing.

B. Server Component Requirements

The following is the itemized requirement for the client component.

1. The server may be executed on a non-ARM11 platform with more computational power.
2. The server must be communicated over the cloud.
3. The server should provide both storage and COD analysis capabilities.
4. The server should implement ANN-based COD analysis.

5. The server should provide the capability of testing ANN for COD analysis.

C. *ANN Model Generator Component Requirements*

The following is the itemized requirement for the client component:

1. The model generator may be executed on a non-ARM11 platform.
2. The ANN model generator should generate a pluggable C-model for the COD analysis server.

III. ARCHITECTURE OF THE SOFTWARE

The overall system is depicted in Fig. 50. The system consists of the following components:

1. The photometer is used to generate the absorption spectrum of the water sample.
2. Cloud storage is used to store the collected absorption spectrum.
3. The cloud server is the application server to analyze the collected absorption spectrum.
4. An ARM11-based portable system is a portable device to collect the spectrum data from the photometer, store the data over cloud storage, and display the analysis results from the cloud server.

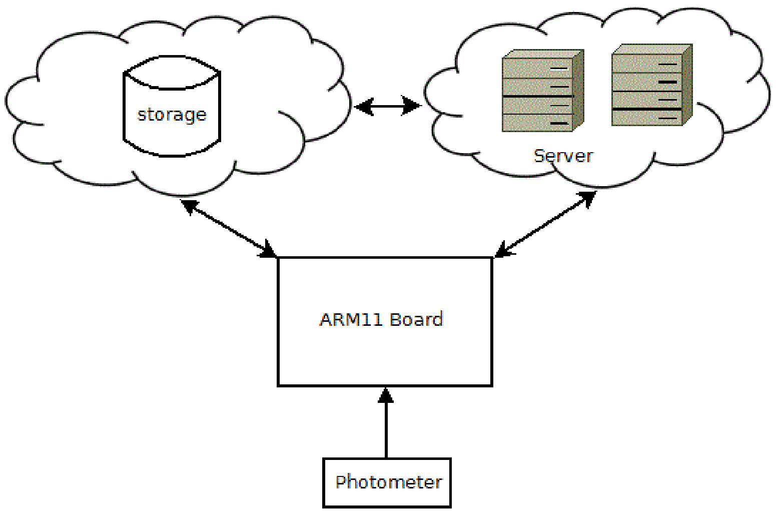


Fig. 50. COD Analysis System.

The software, anCod, is a collection of three distinct utilities as proposed by the requirement described in section II.

1. anCod is the client program, which is primarily executed on the portable ARM11 system.
2. anCodServer is the application server program, which is primarily executed on the cloud server platform.
3. anCodTrainer is the ANN trainer to recognize the COD absorption pattern. This program is also mainly executed on the cloud server platform.

The anCod client software works as the following flow steps depicted in Fig. 51. First it collects photometry data and stores them on the cloud. Second it initiates the analysis on the cloud server, once the data are collected and the deposit is completed. The client would then wait for the server notification of analysis completion and display the analysis results on the local display.

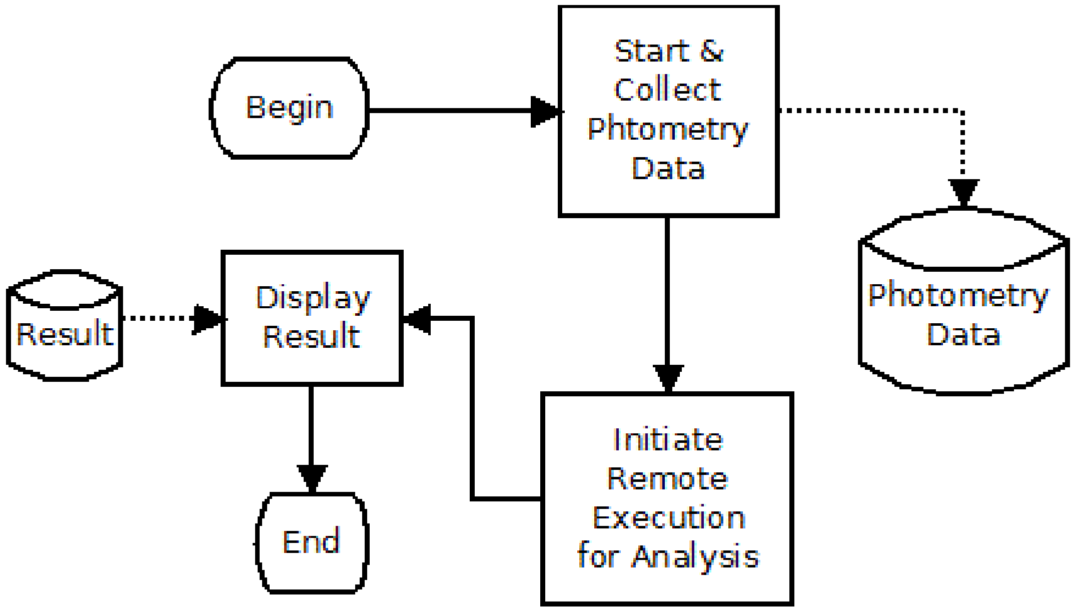

Begin
Start & Collect Phtometry Data
Photometry Data
Result
Display Result
Initiate Remote Execution for Analysis
End

Fig. 51. anCod Client Flow

After receiving the processing request from a client, the server, anCodServer, first preprocesses the collected photometry data compatible with the ANN. This processed information is then analyzed by ANN to produce the COD result for the given water sample. Fig. 52 depicts the overall server-side flow.

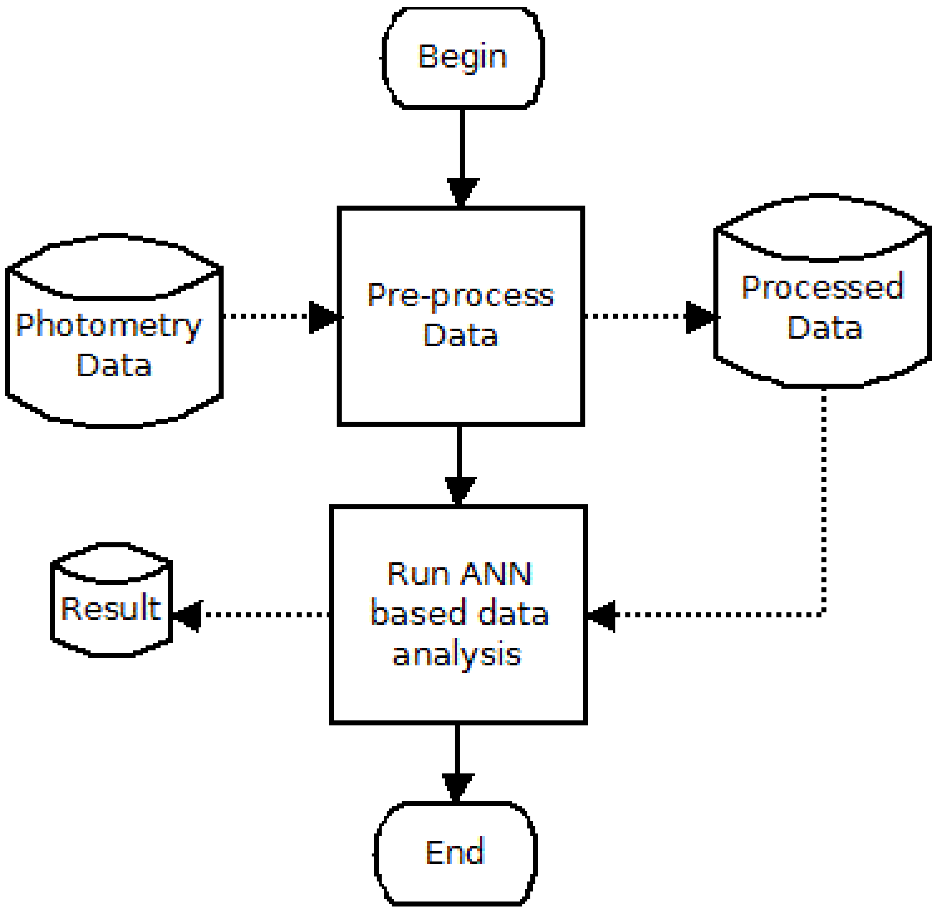


Fig. 52. anCodServer Server Flow

The trainer, anCodTrainer, works on training data and creates an ANN model to be consumed by the anCodServer. The following is the workflow for the anCodTrainer. Fig. 53 depicts the trainer flow.

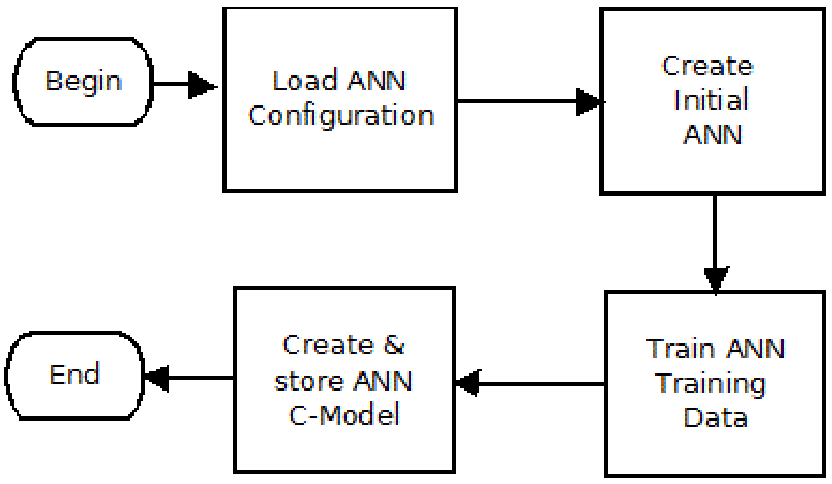


Fig. 53. anCodTrainer Trainer Flow

IV. User Interface

The anCod software suite has two types of user interfaces to control software behavior. One is the command line interface and the other is the parameter setting in the configuration file. In the following subsections, both the interfaces are discussed.

A. Command-line interface

The anCod client software has the following command line usage.

```
$> anCod -help
USAGE : anCod [-sim <sim data file>]
              [-gen_rnd_sim]
              [-train <output value>]
              [-save_as <file name>]
              [-local]
              [-train_sin]
              [-help]
```

Table 28 explains the options. Without any option, anCod captures photometry data from the photometer and requests the server to analyze the data to determine the water COD.

Table 28. The anCod Options

Option	Purpose
-sim <sim data file>	To use photometry data collected off-line or generated with simulation.
-gen_rnd_sim	To generate the ANN training file with random data.
-train <value>	To supply ANN output values for the ANN training file. This option also turns on the training file generation (which keeps appending new training patterns onto the existing training file).
-save_as <file name>	To save the photometry raw data.
-train_sin	To generate ANN training data and test data.
-local	To run analysis locally instead of on the server.
-help	To print on-screen help/usage.

The anCod server software, or anCodServer, has the following command line usage:

```
$> anCodServer -help
USAGE : anCodServer [-test <id>] [-help]
```

Table 29 explains the options for anCodServer. Without any option, anCodServer analyzes the captured photometry data stored by the anCod client and stores the analysis result on a shared remote disk for the anCod client to pick up.

Table 29. The anCodServer Option

Option	Purpose
-test <id>	To run the ANN test for the given ANN type. For the time being, sin and pm are supported. The ID “sin” means to test if ANN recognizes that a signal consists of multiple sine waves. The pm mode test is to

Option	Purpose
	test ANN against the training patterns.
-help	To print on-screen help/usage.

The anCod trainer software, or anCodTrainer, has the following command line usage:

```
$> anCodTrainer -help
USAGE : anCodTrainer -train <id> [-help]
```

Table 30 explains the options for ancodTrainer.

Table 30. The anCodTrainer Options

Option	Purpose
-train <id>	To generate a trained ANN model for a given type of signal recognition. Currently sin and pm IDs are supported. The ID “sin” means to generate an ANN model that recognizes that a signal consists of multiple sine waves. The pm means to generate an ANN model for recognizing real photometry data.
-help	To print on-screen help/usage.

B. Configuration parameters

The anCod software suite uses makefile configuration parameters for the software run. All these parameters are defined as shell (SH) environment variables and have some default values. Users can point to their own configurations using the pointer in the environment variable as shown in the following example:

```
$> setenv ANCOD_USER_CONFIG_FILE ./test_06-config.sh
```

This configuration file is an SH shell script file. Hence, any SH-supported command can be used in this configuration file. The usual command is to set the parameter as the environment variable inside the Bourne shell as follows:

```
export anCod_result_file="anCod_cod.rpt"
```

There are three major categories of parameters:

1. General parameters.
2. Composite sine waves recognizing ANN-related parameters.
3. Composite sine wave-generation parameters.

Table 31 shows the list of available general parameters.

Table 31. List of general parameters for anCod

Parameter	Default	Notes
anCod_data_dir	/export/cod_ann_fs/data	Defines the data directory where the collected photometry data are stored.
anCod_freq_template	anCod_freq.template	Defines the list of frequencies (the x-axis values) used for the collected photometry data.
anCod_data_file	anCod_pm.dat	Name of the data file where the photometry data are collected.
anCod_train_file	anCod_ann_train.dat	Name of the ANN training file generated.
anCod_rsa_private_key	id_rsa.i386 id_rsa.arm11	Name of the private key file used by the remote shell.

Parameter	Default	Notes
anCod_remote_shell_cmd	ssh for i386 dbclient for ARM11	The remote shell name.
anCod_remote_server	boson-srvr-02	Remote server name.
anCod_remote_user	kpatra	Remote user name.
anCod_ model_name	ann_sin	ANN model name.
anCod_model_version	1.0.0	ANN model revision.
anCod_build_tool	/usr/bin/make	Build tool for ANN model generation.
anCod_model_loc	<installation_dir>/ann_ model/<platform>	Directory location where the generated ANN model is stored.
anCod_ann_input	20	Number of input nodes for the ANN.
anCod_ann_output	1	Number of output nodes for the ANN.
anCod_ann_hidden_node	50	Number of nodes in the hidden layer.
anCod_pm_integration_time	2000000	Photometer integration time in microseconds.
anCod_pm_scan_average	10	Amount of data scanned to average out the result from the photometer.
anCod_pm_spec_length	2048	Resolution of photometer operation.
anCod_ref_blank_data	pm-blank.data	Reference photometer data in blank condition. This is used to measure absorption.

Table 32 shows the composite sine recognizing ANN-related parameters. This ANN uses a single hidden layer. The model is generated as lib<model name>_<model version>.so.

Table 32. ANN parameters for composite sine wave recognition

Parameter	Default	Notes
anCod_train_sin_ann_input	20	Number of input nodes for the ANN.
anCod_train_sin_ann_output	1	Number of output nodes for the ANN.
anCod_train_sin_ann_hidden_node	50	Number of nodes in the hidden layer.

Table 33 lists the composite sine wave and corresponding parameters to generate training or test data.

Table 33. Parameters to Generate Train/Test Data for Composite Sine Wave

Parameter	Default	Notes
anCod_train_sin_base_amplitude	1.0	Base amplitude for the composite sine signal.
anCod_train_sin_base_frequency	1000.0	Base frequency in Hz for the composite sine signal.
anCod_train_sin_relative_noise	0.2	Relative random noise amplitude relative to the base amplitude.
anCod_train_sin_sample_point	20	Number of sampling points for ANN training and test data preparation.
anCod_train_sin_no_of_pts	1000	Number of discrete points for the generated composite sine signal.

Parameter	Default	Notes
anCod_train_sin_harmonics_amps	1.0	Comma-separated relative amplitude values for the harmonics in the composite sine signals. Example: 1.0,0.8,0.4,0.2,0.1,0.05,0.25,0.25,0.05,0.1,0.2,0.4,0.8,1.0
anCod_train_sin_phase_shift	true	Adds random phase shift per composite sine signal generation. The shift happens randomly between 0-PI if set to true.
anCod_train_sin_include_random_signal	true	Generates random noise as training pattern that results in nonrecognition of the composite sine signal.
anCod_train_sin_train_file	anCod_train_sin.train	Generated ANN training file name.
anCod_train_sin_data_file	anCod_train_sin.dat	Data file base name for each composite sine signal used in ANN training. The actual file name is extended with <rnd|sin>_train_<unique id>
anCod_test_sin_data_file	anCod_test_sin.dat	The data file base name for each composite sine signal. The actual file name is extended with <rnd|sin>_test_<unique id>
anCod_train_sin_test_file	anCod_train_sin.test	Generated ANN test file name.
anCod_train_sin_no_of_noise_data	100	Amount of training/test data to be generated with random noise.
anCod_train_sin_no_of_amp_data	1	Level of amplitude variation. The base amplitude is multiplied by the current iteration value.

Parameter	Default	Notes
anCod_train_sin_no_of_freq_data	1	Level of frequency variation. The base frequency is multiplied with the current iteration value.
anCod_generate_data_file	false	If set to true, all the signal patterns used in ANN training/test file generation are stored in the data directory.

V. IMPLEMENTATION DETAILS

For the initial implementation, a traditional NFS-based file system is used in place of cloud-based storage. Likewise, a secured remote shell (SSH)-based application execution is used instead of a cloud-based application execution. These basic changes should be sufficient to prove the concept of this software because the services using the cloud are accounted remote storage and servers. With these basic changes, the top-level architecture looks like Fig. 54.

The top-level commands—anCod, anCodServer, and anCodTrainer—are shell scripts that load all the required configuration parameters, determine the execution platform, and invoke the corresponding binary executable for the execution platform. This software can be built for both i386 and ARM11 architecture. Thus, the software can be executed on both platforms.

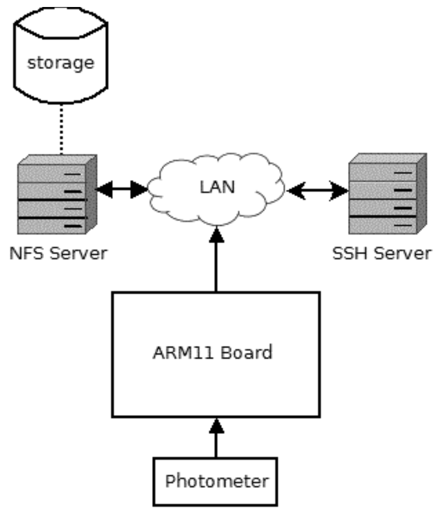


Fig. 54. Implemented Software Architecture

Table 34 shows the available targets that developers may invoke in a make-based build call.

Table 34. Makefile Targets to Build an anCod Application

Target	Purpose
all	To build and install for both i386 and ARM11.
build	To build the software.
install	To install the software.
smoke	To smoke test the software. Each command will be invoked with the "-help" option.
clean	To clean the temporary build file for the given platform.
clean_all	To clean the temporary build files for the entire platform.

Each build behavior can be controlled through the macros shown in Table 35. The software installed in the directory is determined as follows: $(ANCOD_ROOT)/anCod/$(VERSION).

Table 35. Makefile Macros Used in Building the anCod Application

Macro	Description
ANCOD_ROOT	Root of software installation
VERSION	Software revision
ARCH	Target platform for the software (i386 or arm11)

The source directory root contains the following build files as shown in Table 36.

Table 36. List of Makefiles for anCod

File name	Notes
Makefile	Top-level makefile.
MakeFiles/Common.mk	Common macro definitions for all the modules.
MakeFiles/Rules.mk	All the build rules are defined here at this makefile.
src/<module>/Makefile	Source files and build related definitions for the module.

The makefile for each module should contain the following macros, as shown in order in Table 37, to define build-related definitions and source files.

Table 37. Macros for Module Makefile in anCod Software

Macro	Description
TOP_DIR	Directory location for the MakeFiles dir.
MODULE_NAME	Name of the module. All the following macros are related to this value.
$(MODULE_NAME)-SRC	List of source code to be compiled.
$(MODULE_NAME)-INCLUDE	Include path definition for compilation.
$(MODULE_NAME)-LIBS	Linking flag definitions.
$(MODULE_NAME)-EXE	Generated executable name.

The MakeFiles/Common.mk and the MakeFiles/Rules.mk should be included in the module makefile at a proper place as in the following example.

```
TOP_DIR = $(shell cd ../../; pwd)
MODULE_NAME = trainer
# include common make files
include $(TOP_DIR)/MakeFiles/Common.mk
trainer-SRC     = anCodTrainer_main.c \
                  anCodTrainer_sin.c \
                  anCodTrainer_1_layer_ann.c \
                  anCodTrainer_multilayer_feedforward_ann.c \
                  anCodTrainer_build_model.c
trainer-INCLUDE = $(FANN_INCLUDE)
trainer-LIBS    = $(FANN_LIB) $(COMMON_LIB_FLAG)
trainer-EXE     = anCodTrainer
include $(TOP_DIR)/MakeFiles/Rules.mk
```

VI. EXPERIMENT RESULTS

For composite sine signal recognition, an experiment with the following attributes was performed:

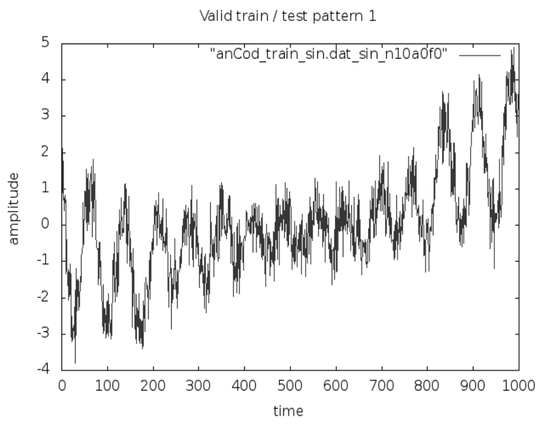
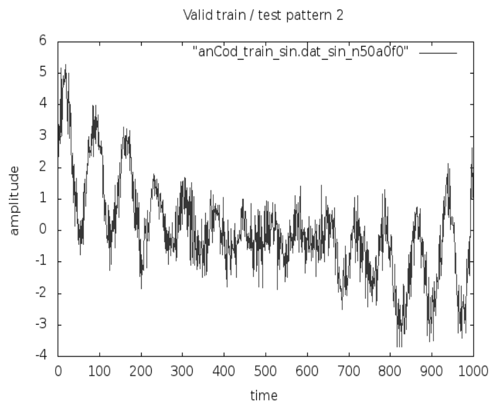
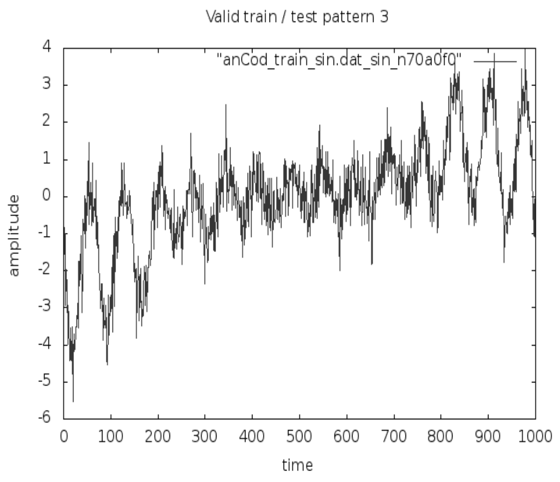
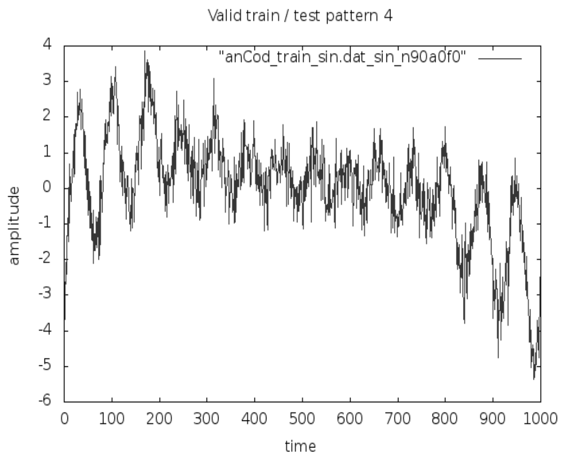
1. The composite sine signal was generated with a relative amplitude of component frequencies as {1.0,0.8,0.4,0.2,0.1,0.05,0.25,0.25,0.05,0.1,0.2,0.4,0.8,1.0}.
2. One hundred training patterns with a composite sine signal and 100 training patterns with random white noise were created.

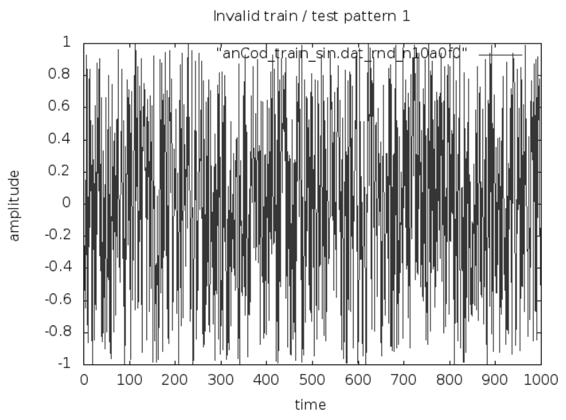
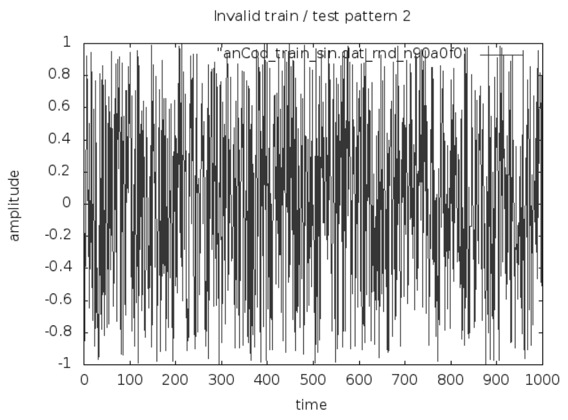
3. One hundred test patterns with a composite sine signal and 100 test patterns with random white noise were created.

4. The ANN used 20 inputs, one output, and 50 hidden nodes in a single perception layer.

Table 38 shows some examples of valid and invalid patterns, which were used to train and test the ANN.

Table 38. anCod Validation Test Patterns

Train/test pattern	Train/test pattern
	
	

Train/test pattern	Train/test pattern
	

The test results are as follows:

```
$>./runme.test
INFO : Using user configuration file ./test_06-config.sh ...
****************************************************
                   annCodServer (1.0.0)
****************************************************
INSTALL DIR          : /import/thesis/Study/Thesis/COD-ANN/ancod-
install/anCod
DATA DIR             : ./data
FREQ TEMPLATE        : anCod_freq.template
DATA FILE            : anCod_pm.dat
RESULT FILE          : anCod_cod.rpt
EXECUTED ON          : boson-lnx-01 (i386)
EXECUTED ON          : Fri Jan 25 18:11:12 PST 2013
CMD ARG              : -test sin
****************************************************
INFO [anCodServer_sin.c@35)]: Using test file
./data/anCod_train_sin.test
INFO [anCodServer_sin.c@39)]: ANN model ann_test_sin version 1.0.1 is
used from dir /import/thesis/Study/Thesis/COD-
ANN/anCod/test/test_anCod_06
INFO [anCodServer_sin.c@76)]: 0 out of 200 tests failed (0.000000%)
```

The tests results show that all the valid patterns were recognized correctly and all the invalid patterns were discarded correctly.

VII. Conclusion

The test results show that the software anCod is ready to be used in the proposed thesis.