Sensor Interfacing in Marine Instrumentation

Report on Summer Internship

Under the guidance of

Shri. Sanjeev Afzulpurkar

Chief Scientist
Marine Instrumentation Division
CSIR- National Institute of Oceanography
Dona Paula, Goa

By

Mirza Ali Beg

Roll Number: 141104029
4th Semester, Bachelor of Engineering
Department of Electronics and Telecommunication
Goa College of Engineering, Goa

July 2016

"This Report is submitted as a part of curriculum for Bachelor of Engineering Programme at Goa College of Engineering, Goa"

CERTIFICATION

This is to certify that this internship is a bonafide and an authentic record of the study and research carried out by Mirza Ali Beg, student of B.E. Electronics and Telecommunication Engineering (E.T.C, Roll number 141104029) from Goa College of Engineering, under my supervision and guidance at CSIR - National Institute of Oceanography, Dona Paula, Goa. No part of this Internship has been presented before for any degree or diploma in any University.

[illegible]

6th September 2016

(Sanjeev Afzulpurkar)

Chief Scientist
CSIR - National Institute of Oceanography
Dona Paula
Goa

1. Acknowledgment

I would like to express the deepest gratitude and highest regard for my project guide Shri. Sanjeev Afzulpurkar for being the driving force behind this project. It was through his motivation and guidance that I was able to carry out this project.

I would like to thank Dr. H. G. Virani, Head of Electronics and Telecommunication Department of Goa College of Engineering and also all my teachers for bestowing me with the knowledge to carry out this project.

I am grateful to Dr. S. W. A Naqvi, Director of CSIR - National Institute of Oceanography for giving me the opportunity to carry out this project as my summer internship in this reputed institute. This was a great exposure for me to understand how actual research and development is done in such prestigious institutes.

1. Introduction

In the broadest definition, a sensor is an object whose purpose is to detect events or changes in its environment, and then provide a corresponding output. A sensor is a type of transducer; sensors may provide various types of output, but typically use electrical or optical signals.

2.1 Oceanographic Sensors

There are various sensors used underwater for the measurement of various parameters like temperature, pressure, depth, current velocity and so on. A few of the most common ones used are:

2.1.1 CTDs:

Also called Sonde is an oceanography instrument used to determine the conductivity, temperature, and depth of the ocean (Hence CTD). The CTD may be incorporated into an array of Niskin bottles referred to as a carousel or rosette. The sampling bottles close at predefined depths, triggered either manually or by a computer, and the water samples may subsequently be analyzed further for biological and chemical parameters.

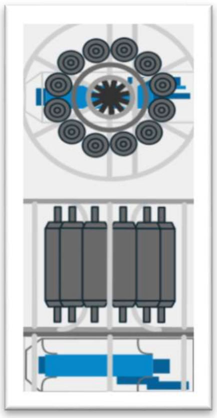


Figure 1- CTD Equipment with Nisken bottles & logger

The instrument is a cluster of sensors which measure conductivity, temperature, and pressure. Sensors commonly scan at a rate of 24 Hz. Depth measurements are derived from measurement of hydrostatic pressure, and salinity is measured from electrical conductivity. Sensors are arranged inside a metal housing, the metal used for the housing determining the depth to which the CTD can be lowered. Titanium housings allow sampling to depths in excess of 10,000 metres. Other sensors may be added to the cluster, including some that measure chemical or biological parameters, such as dissolved oxygen and chlorophyll fluorescence, the latter an indication of the concentration of microscopic photosynthetic organisms (phytoplankton) contained in the water.

The advantage of CTD casts is the acquisition of high resolution data. The limitation of CTD sampling is that only a point in space (the sampling site) can be sampled at one time, and many casts, which are costly and time-consuming, are needed to acquire a broad picture of the marine environment of interest.

We will be using CTDs as an example in the tests that we will conduct later, specifically CTDs like Glider Payload CTD by Sea-Bird Scientific and Fast CTD Profiler by Valeport

2.1.2 Acoustic Doppler Current Profiler(ADCP):

It is a hydro-acoustic current meter similar to a sonar, attempting to measure water current velocities over a depth range using the Doppler effect of sound waves scattered back from particles within the water column. The term ADCP is a generic term for all acoustic current profilers[1] although the abbreviation originates from an instrument series introduced by RD Instruments in the 1980s. The working frequencies range of ADCPs range from 38 kHz to several Megahertz.

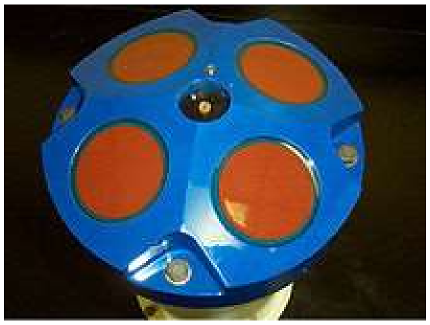


Figure 2- Head of an ADCP with four transducers (Model WH-600, RD Instruments)

ADCPs contain piezoelectric transducers to transmit and receive sound signals. The traveling time of sound waves gives an estimate of the distance. The frequency shift of the echo is proportional to the water velocity along the acoustic path. To measure 3D velocities, at least three beams are required. In rivers, only the 2D velocity is relevant and ADCPs typically have two beams. In recent years, more functionality has been added to ADCPs (notably wave and turbulence measurements) and systems can be found with 2,3,4,5 or even 9 beams.

The various applications of ADCPs are Bottom Tracking- By adjusting the window where the Doppler shift is calculated, it is possible to measure the relative velocity between the instrument and the bottom. This feature is referred to as bottom-track.

Discharge Measurements- Here, the ADCP is used to measure the total water transport across the cross-sectional area of a river. Doppler Velocity Log(DVL)- It is used for estimating positions of underwater vehicles and for their navigation.

Wave Measurements- To measure height and direction of the waves. Turbulence- To measure turbulent parameters used mostly by stationary deployments

2.1.3 Attitude and Heading Reference Systems(AHRS):

This system consists of sensors on three axes that provide attitude information for aircraft, including roll, pitch and yaw. They are designed to replace traditional mechanical gyroscopic flight instruments and provide superior reliability and accuracy AHRS consist of either solid-state or microelectromechanical systems (MEMS) gyroscopes, accelerometers and magnetometers on all three axes. The key difference between an inertial measurement unit (IMU) and an AHRS is the addition of an on-board processing system in an AHRS which provides attitude and heading information versus an IMU which just delivers sensor data to an additional device that computes attitude and heading. In addition to attitude determination an AHRS may also form part of an inertial navigation system.

2.1.4 Bathythermograph(BT):

It is a small torpedo-shaped device that holds a temperature sensor and a transducer to detect changes in water temperature versus depth down to a depth of approximately 285 meters (935 feet). Lowered by a small winch on the ship into the water, the BT records pressure and temperature changes on a coated glass slide as it is dropped nearly freely through the water. While the instrument is being dropped, the wire is paid out until it reaches a predetermined depth, then a brake is applied and the BT is drawn back to the surface. Because the pressure is a function of depth, temperature measurements can be correlated with the depth at which they are recorded.

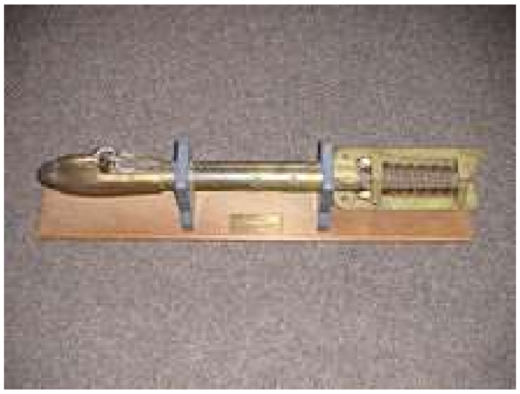


Figure 3- A bathythermograph

2.2 Different Platforms used for interfacing these sensors

2.2.1 Serial port:

In computing, a serial port is a serial communication interface through which information transfers in or out one bit at a time (in contrast to a parallel port). Throughout most of the history of personal computers, data was transferred through serial ports to devices such as modems, terminals and various peripherals.

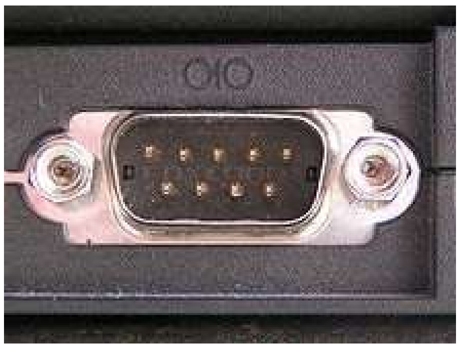


Figure 4- Male DE-9 connector used for a serial port on an IBM PC

While such interfaces as Ethernet, FireWire, and USB all send data as a serial stream, the term "serial port" usually identifies hardware more or less compliant to the RS-232 standard, intended to interface with a modem or with a similar communication device.

Modern computers without serial ports may require serial-to-USB converters to allow compatibility with RS 232 serial devices. Serial ports are still used in applications such as

industrial automation systems, scientific instruments and sensors, point of sale systems and some industrial and consumer products. Server computers may use a serial port as a control console for diagnostics. Network equipment (such as routers and switches) often use serial console for configuration. Serial ports are still used in these areas as they are simple, cheap and their console functions are highly standardized and widespread. A serial port requires very little supporting software from the host system.
Majority of the sensors are compatible with serial ports as it is an old and widely used technology. It is considered to be the primary platform for interfacing others being secondary since some sensors may not support them.

2.2.2 USB:

USB, short for Universal Serial Bus, is an industry standard developed in the mid-1990s that defines the cables, connectors and communications protocols used in a bus for connection, communication, and power supply between computers and electronic devices. It is currently developed by the USB Implementers Forum (USB IF).

Figure 5- USB logo on the head of a standard A plug, the most common USB plug

USB was designed to standardize the connection of computer peripherals (including keyboards, pointing devices, digital cameras, printers, portable media players, disk drives and network adapters) to personal computers, both to communicate and to supply electric power. It has become commonplace on other devices, such as smartphones, PDAs and video game consoles. USB has effectively replaced a variety of earlier interfaces, such as parallel ports, as well as separate power chargers for portable devices.
USB is the easiest platform for interfacing although it may not be supported by some sensors in the market.

2.2.3 Ethernet:

Ethernet is a family of computer networking technologies commonly used in local area networks (LANs) and metropolitan area networks (MANs). It was commercially introduced in 1980 and first standardized in 1983 as IEEE 802.3, and has since been refined to support higher bit rates and longer link distances. Over time, Ethernet has largely replaced competing wired LAN technologies such as token ring, FDDI and ARCNET.

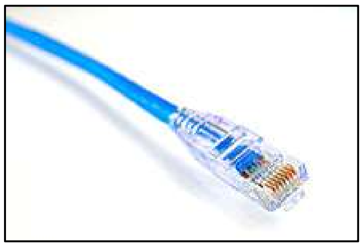


Figure 6- 8P8C modular connector (often called RJ45)

The original 10BASE5 Ethernet uses coaxial cable as a shared medium, while the newer Ethernet variants use twisted pair and fiber optic links in conjunction with hubs or switches. Over the course of its history, Ethernet data transfer rates have been increased from the original 2.94 megabits per second (Mbit/s) to the latest 100 gigabits per second (Gbit/s), with 400 Gbit/s expected by late 2017. The

Ethernet standards comprise several wiring and signaling variants of the OSI physical layer in use with Ethernet.

Systems communicating over Ethernet divide a stream of data into shorter pieces called frames. Each frame contains source and destination addresses, and error-checking data so that damaged frames can be detected and discarded; most often, higher-layer protocols trigger retransmission of lost frames. As per the OSI model, Ethernet provides services up to and including the data link layer.

Since its commercial release, Ethernet has retained a good degree of backward compatibility. Features such as the 48-bit MAC address and Ethernet frame format have influenced other networking protocols. The primary alternative for some uses of contemporary LANs is Wi-Fi, a wireless protocol standardized as IEEE 802.11.
Ethernet has replaced InfiniBand as the most popular system interconnect of TOP500 supercomputers.

The most commonly used Ethernet cable is the RJ-45, this platform provides very high speed interfacing and communication but it is not supported by many sensors

3. Software Used

3.1 Terminal v1.91b by Br@y++

Terminal is a simple serial port (COM) terminal emulation program. It can be used for communication with different devices such as modems, routers, embedded uC systems, GSM phones, GPS modules... It is very useful debugging tool for serial communication applications.

3.2 Virtual Serial Port Driver 6.9 by Eltima Software

Virtual Serial Port Driver creates virtual serial ports and connects them in pairs via virtual null modem cable. Applications on both ends of the pair will be able to exchange data in such a way, that everything written to the first port will appear in the second one and backwards. All virtual serial ports work and behave exactly like real ones, emulating all their settings. You can create as many virtual port pairs as you want, so there will be no serial ports shortage and no additional hardware crowding your desk.

3.3 MATLAB R2015a

MATLAB stands for matrix laboratory and it is a high-performance language for technical computing. It integrates computation, visualization, and programming in an easy-to-use environment where problems and solutions are expressed in familiar mathematical notation. Typical uses include:

- Math and computation
- Algorithm development
- Modeling, simulation, and prototyping
- Data analysis, exploration, and visualization
- Scientific and engineering graphics
- Application development, including Graphical User Interface building

4. Objective

To make a sensor, universal plug-n-play, i.e. To make an interface with the help of an embedded system and a computer which can read any sensor when connected to a Serial RS-232 port and display the type, model, manufacturer and the output data given by the sensor in a common format. This process would require detecting the active COM ports, matching the baud rates of the sensor and the interface, then analyzing the raw output data acquired with the help of known formats specified for each different sensor and then finally displaying the output from the sensor in the desired format.

Hence, broadly we need to accomplish the following-

4.1 Sensor Detection
4.2 Data Acquisition
4.3 Data Analysis
4.4 Data Storage

Here we shall primarily concentrate on the Sensor Detection part of this project.

5. Procedure

For this entire process of sensor detection, we require four main components: -

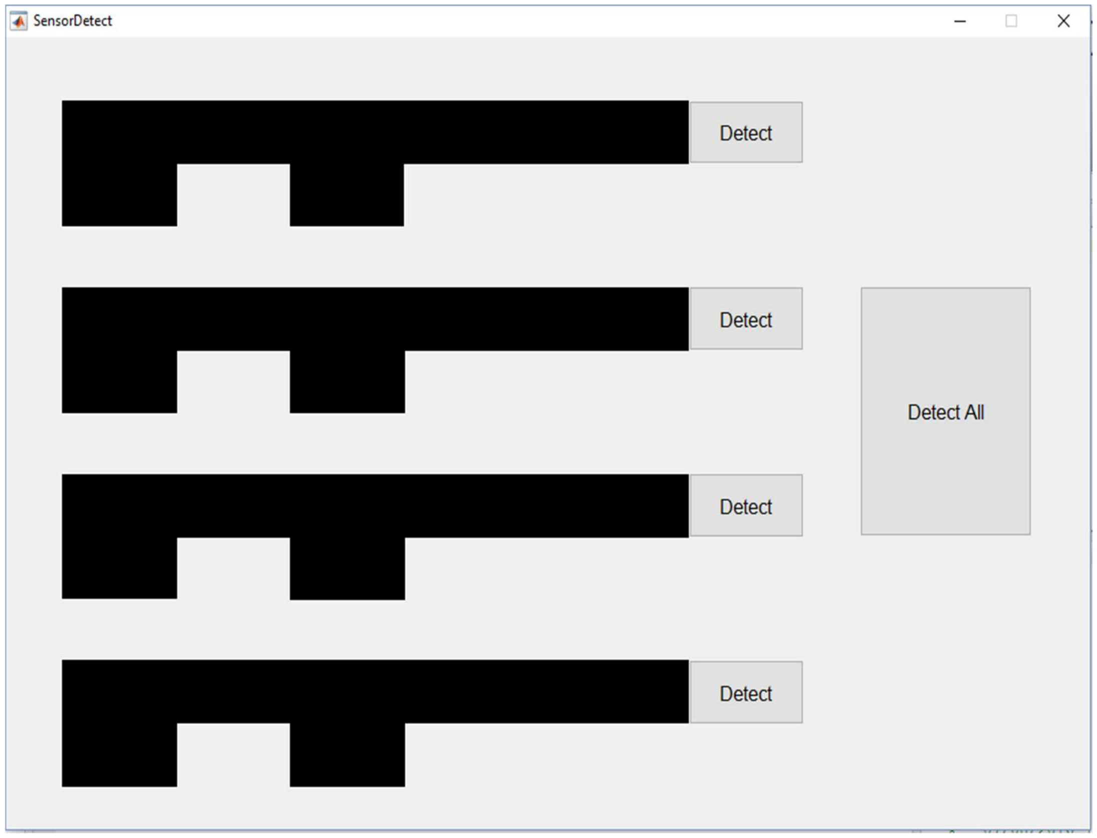
- A Computer/ Laptop with intel core i3 with minimum four gigabytes of RAM capable of running MATLAB R2015a
- An Arduino board
- Any Oceanographic Sensor with RS-232 serial connection support
- RS-232 Serial Cables (USB to RS232 or double ended RS232 depending on the ports available)

Due to limitations of the resources available we will simulate the work of the Arduino, Sensor and the RS-232 serial cables with the help of software such as

- MATLAB R2015a, which can integrate the work of an Arduino board for the purpose of simulations
- Terminal v1.91b by br@y++, that can transmit/ Receive data through serial ports similar to a sensor
- Virtual Serial Port Driver 6.9 by Eltima Software (VSPD), which can create virtual serial ports that can be used to transmit and receive data just like actual serial ports

In MATLAB we develop a program using the 'guide' menu to create a graphical user interface (GUI) that would display the power ON message transmitted by a sensor (terminal) i.e. information such as sensor type, model, manufacturer etc.

The GUI would look like this:



The MATLAB code written for this program as shown in the later section will allow us to detect up to four sensors individually or simultaneously using the "Detect All" button. This can further be replicated to n number of sensors.

On the Initialization of the program, certain variables will be defined most importantly two arrays 'COM' and 'bdr' where the former will store the names of the COM ports available at the time and the latter will store values of the possible baud rates of the serial communication.

Once we press a detect button two loops, one inside the other will start that would check for data received by each serial port available and at each possible baud rate.

If we receive the Information message at power ON of the sensor, then that message will be directly displayed along with the number of the COM port and the baud rate of the communication channel.

If no data is received, then the program will initiate a 'handshake' by transmitting a single ASCII character 'A' and wait for a reply. If we get a reply, then that means the connection has been established and a message will be displayed for the same. But if we do not get any reply then that means the connection is not proper.

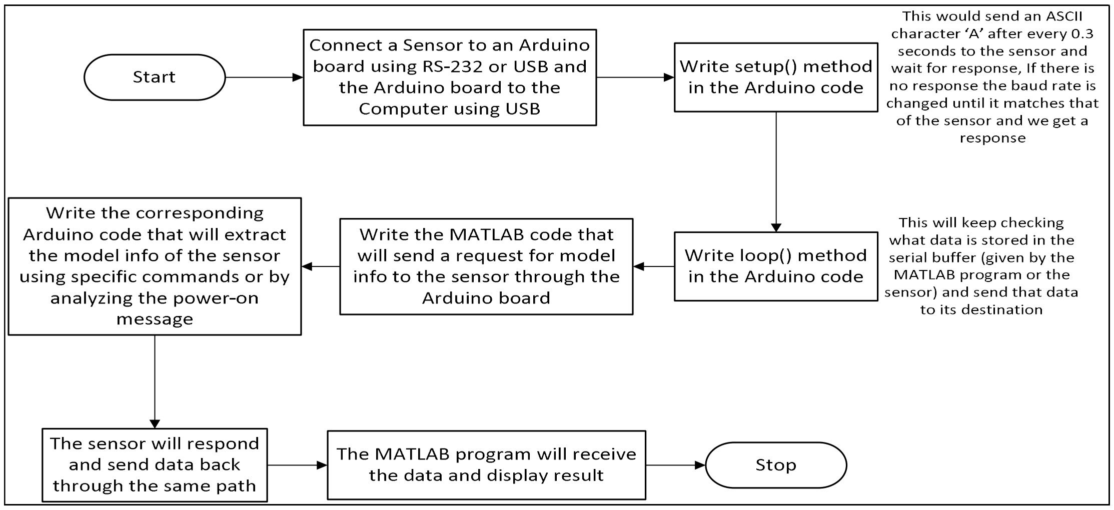
Along with the displaying action the MATLAB program will transmit a command requesting for the format of the output data from the sensor. Once the sensor gives the format, this format will be stored in an external text file "Data.txt". The same text file will also store the various readings collected by the sensor during operation.

If multiple sensors are detected simultaneously then the text file will first store the format of each sensor and then it will start storing the sensor readings.

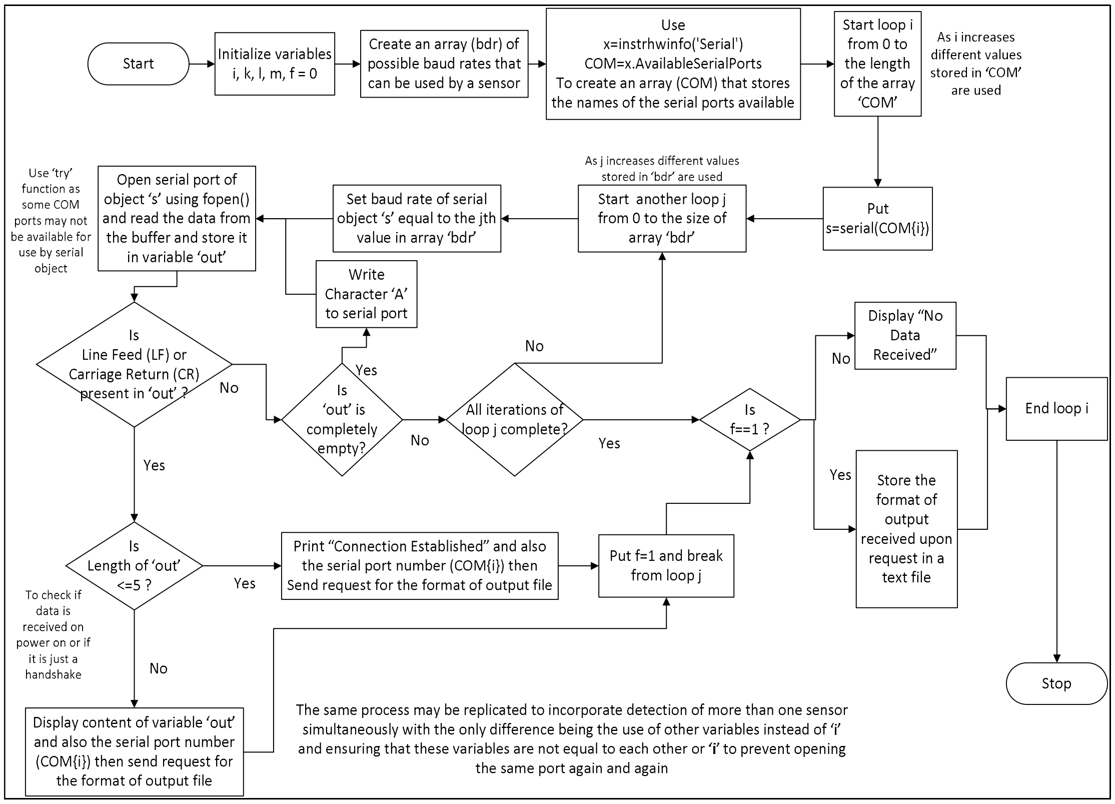
This will finally complete the sensor detection process.

6. Flowcharts Explaining the Logic

6.1 Generalized Flowchart of the logic used for sensor detection


Start
Connect a Sensor to an Arduino board using RS-232 or USB and the Arduino board to the Computer using USB
Write setup() method in the Arduino code
This would send an ASCII character 'A' after every 0.3 seconds to the sensor and wait for response, If there is no response the baud rate is changed until it matches that of the sensor and we get a response
Write loop() method in the Arduino code
This will keep checking what data is stored in the serial buffer (given by the MATLAB program or the sensor) and send that data to its destination
Write the MATLAB code that will send a request for model info to the sensor through the Arduino board
Write the corresponding Arduino code that will extract the model info of the sensor using specific commands or by analyzing the power-on message
The sensor will respond and send data back through the same path
The MATLAB program will receive the data and display result
Stop

6.2 Flowchart explaining the logic used in the MATLAB program for detection of sensors


Start
Initialize variables i, k, l, m, f = 0
Create an array (bdr) of possible baud rates that can be used by a sensor
Use x=instrhwinfo('Serial') COM=x.AvailableSerialPorts To create an array (COM) that stores the names of the serial ports available
Start loop i from 0 to the length of the array 'COM'
As i increases different values stored in 'COM' are used
Put s=serial(COM{i})
As j increases different values stored in 'bdr' are used
Start another loop j from 0 to the size of array 'bdr'
Set baud rate of serial object 's' equal to the jth value in array 'bdr'
Open serial port of object 's' using fopen() and read the data from the buffer and store it in variable 'out'
Use 'try' function as some COM ports may not be available for use by serial object
Write Character 'A' to serial port
Is Line Feed (LF) or Carriage Return (CR) present in 'out' ?
No
Yes
Is 'out' is completely empty?
Yes
No
All iterations of loop j complete?
No
Yes
Is f==1 ?
No
Display "No Data Received"
Yes
Store the format of output received upon request in a text file
End loop i
Stop
Is Length of 'out' <=5 ?
Yes
No
To check if data is received on power on or if it is just a handshake
Print "Connection Established" and also the serial port number (COM{i}) then Send request for the format of output file
Put f=1 and break from loop j
Display content of variable 'out' and also the serial port number (COM{i}) then send request for the format of output file
The same process may be replicated to incorporate detection of more than one sensor simultaneously with the only difference being the use of other variables instead of 'i' and ensuring that these variables are not equal to each other or 'i' to prevent opening the same port again and again

7. MATLAB Code for Sensor Detection

```
----------------------------------------------------------------------------------------------------------------------------
----

function varargout = SensorDetect(varargin)
% SENSORDETECT MATLAB code for SensorDetect.fig
%      SENSORDETECT, by itself, creates a new SENSORDETECT or raises the existing
%      singleton*.
%
%      H = SENSORDETECT returns the handle to a new SENSORDETECT or the handle to
%      the existing singleton*.
%
%      SENSORDETECT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in SENSORDETECT.M with the given input arguments.
%+
%      SENSORDETECT('Property','Value',...) creates a new SENSORDETECT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before SensorDetect_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to SensorDetect_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help SensorDetect

% Last Modified by GUIDE v2.5 30-Jul-2016 18:28:57

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @SensorDetect_OpeningFcn, ...
                   'gui_OutputFcn',  @SensorDetect_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before SensorDetect is made visible.
function SensorDetect_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to SensorDetect (see VARARGIN)

% Choose default command line output for SensorDetect
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes SensorDetect wait for user response (see UIRESUME)
% uiwait(handles.figure1);
% --- Outputs from this function are returned to the command line.
function varargout = SensorDetect_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes during object creation, after setting all properties.
function Recieve1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Recieve1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called
global i k l m;
i=0;
k=0;
l=0;
m=0;

% --- Executes on button press in Detect1.
function Detect1_Callback(hObject, eventdata, handles)
% hObject    handle to Detect1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global outa outa1 i bA comA;
bdr=[9600 14400 19200 28800 38400 57600 115200];
x=instrhwinfo('Serial');
f=0;
COM=x.AvailableSerialPorts;
len=length(COM);
for i=1:1:len
    comA=COM{i};
    s=serial(COM{i});
    for j=1:1:7
        bA=bdr(j);
        set(s,'BaudRate',bA,'Terminator','CR');
        try
            fopen(s);
            outa=fscanf(s);
            if isvector(strfind(outa,char(13)))||isvector(strfind(outa,char(10)))
                if length(outa)<=5
                    set(handles.Recieve1,'String','Connection Established...Ready for
input')
                    set(handles.com1,'String',COM{i})
                    set(handles.bd1, 'String', bA)
                    guidata(hObject, handles);
                    fprintf(s,'formatcmd');
                    f=1;
                    break;
                else
                    set(handles.Recieve1,'String',outa)
                    set(handles.com1,'String',COM{i})
                    set(handles.bd1, 'String', bA)
                    guidata(hObject, handles);
                    fprintf(s,'formatcmd');
                    f=1;
                    break;
                end
            else
                if isempty(outa)
                   fprintf(s, 'A');
                end
            end
        end
        clear b;
    end
    if f==1
        while 1
            flushinput(s);
            outa1=fscanf(s);
            if ~strcmp(outa, outa1)
                fid1=fopen('Data.txt','w');
                fprintf(fid1,'%s %s', 'Sensor1 ');
                fprintf(fid1,'%s %s\r\n', outa1);
                fprintf(s,'datacmd');
                break;
            end
        end
        fclose(fid1);
        break;
    else
         set(handles.Recieve1,'String','No Data Recieved, Check your Connection')
         if i==len
             break;
         end
    end
end
if f==0
    i=0;
end
flushinput(s);
flushoutput(s);
try
     fclose(s);
 end

% --- Executes during object creation, after setting all properties.
function Recieve2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Recieve2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called


% --- Executes on button press in Detect2.
function Detect2_Callback(hObject, eventdata, handles)
% hObject    handle to Detect2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global outb outb1 k i bB comB;
bdr=[9600 14400 19200 28800 38400 57600 115200];
x=instrhwinfo('Serial');
f=0;
COM=x.AvailableSerialPorts;
len=length(COM);
for k=1:1:len
    if k~=i
        comB=COM{k};
    d=serial(COM{k});
    for j=1:1:7
        bB=bdr(j);
        set(d,'BaudRate',bB,'Terminator','CR');
        try
            fopen(d);
            outb=fscanf(d);
            if isvector(strfind(outb,char(13)))||isvector(strfind(outb,char(10)))
                if length(outb)<=5
                    set(handles.Recieve2,'String','Connection Established...Ready for
input')
                    set(handles.com2,'String',COM{k})
                    set(handles.bd2, 'String', bB)
                    guidata(hObject, handles);
                    fprintf(d,'formatcmd');
                    f=1;
                    break;
                else
```

```
                    set(handles.Recieve2,'String',outb)
                    set(handles.com2,'String',COM{k})
                    set(handles.bd2, 'String', bB)
                    guidata(hObject, handles);
                    fprintf(d,'formatcmd');
                    f=1;
                    break;
                end
            else
                if isempty(outb)
                    fprintf(d, 'A');
                end
            end
        end
        clear b;
    end
    if f==1
        while 1
            flushinput(d);
            outb1=fscanf(d);
            if ~strcmp(outb, outb1)
                fid2=fopen('Data.txt','a');
                fprintf(fid2,'%s %s', 'Sensor2 ');
                fprintf(fid2,'%s %s\r\n', outb1);
                fprintf(d,'datacmd');
                break;
            end
        end
        fclose(fid2);
        break;
    else
        set(handles.Recieve2,'String','No Data Recieved, Check your Connection')
        if k==len
             break;
        end
    end
   end
end
if f==0
    k=0;
end
flushinput(d);
flushoutput(d);
try
     fclose(d);
 end



% --- Executes on button press in Detect3.
function Detect3_Callback(hObject, eventdata, handles)
% hObject    handle to Detect3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global outc outc1 k i l comC bC;
bdr=[9600 14400 19200 28800 38400 57600 115200];
x=instrhwinfo('Serial');
f=0;
COM=x.AvailableSerialPorts;
len=length(COM);
for l=1:1:len
    if l~=i && l~=k
        comC=COM{l};
    a=serial(COM{l});
    for j=1:1:7
        bC=bdr(j);
        set(a,'BaudRate',bC,'Terminator','CR');
        try
            fopen(a);
            outc=fscanf(a);
            if isvector(strfind(outc,char(13)))||isvector(strfind(outc,char(10)))
                if length(outc)<=5
                    set(handles.Recieve3,'String','Connection Established...Ready for
input')

                    set(handles.com3,'String',COM{l})
                    set(handles.bd3, 'String', bC)
                    guidata(hObject, handles);
                    fprintf(a,'formatcmd');
                    f=1;
                    break;
                else
                    set(handles.Recieve3,'String',outc)
                    set(handles.com3,'String',COM{l})
                    set(handles.bd3, 'String', bC)
                    guidata(hObject, handles);
                    fprintf(a,'formatcmd');
                    f=1;
                    break;
                end
            else
                if isempty(outc)
                    fprintf(a, 'A');
                end
            end
        end
        clear b;
    end
    if f==1
        while 1
            flushinput(a);
            outc1=fscanf(a);
            if ~strcmp(outc, outc1)
                fid3=fopen('Data.txt','a');
                fprintf(fid3,'%s %s', ' Sensor3 ');
                fprintf(fid3,'%s %s\r\n', outc1);
                fprintf(a,'datacmd');
                break;
            end
        end
        fclose(fid3);
        break;
    else
        set(handles.Recieve3,'String','No Data Recieved, Check your Connection')
        if l==len
             break;
        end
    end
    end
end
if f==0
    l=0;
end
flushinput(a);
flushoutput(a);
try
     fclose(a);
 end



% --- Executes on button press in Detect4.
function Detect4_Callback(hObject, eventdata, handles)
% hObject    handle to Detect4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global outd outd1 k i l m comD bD;
bdr=[9600 14400 19200 28800 38400 57600 115200];
x=instrhwinfo('Serial');
f=0;
COM=x.AvailableSerialPorts;
```

```
len=length(COM);
for m=1:1:len
    if m~=i && m~=l && m~=k
        comD=COM{m};
    w=serial(COM{m});
    for j=1:1:7
        bD=bdr(j);
        set(w,'BaudRate',bD,'Terminator','CR');
        try
            fopen(w);
            outd=fscanf(w);
            if isvector(strfind(outd,char(13)))||isvector(strfind(outd,char(10)))
                if length(outd)<=5
                    set(handles.Recieve4,'String','Connection Established...Ready for
input')
                    set(handles.com4,'String',COM{m})
                    set(handles.bd4, 'String', bD)
                    guidata(hObject, handles);
                    fprintf(w,'formatcmd');
                    f=1;
                    break;
                else
                    set(handles.Recieve4,'String',outd)
                    set(handles.com4,'String',COM{m})
                    set(handles.bd4, 'String', bD)
                    guidata(hObject, handles);
                    fprintf(w,'formatcmd');
                    f=1;
                    break;
                end
            else
                if isempty(outd)
                    fprintf(w, 'A');
                end
            end
        end
        clear b;
    end
    if f==1
       while 1
            flushinput(w);
            outd1=fscanf(w);
            if ~strcmp(outd, outd1)
                fid4=fopen('Data.txt','a');
                fprintf(fid4,'%s %s', 'Sensor4 ');
                fprintf(fid4,'%s %s\r\n', outd1);
                fprintf(w,'datacmd')
                break;
            end
        end
        fclose(fid4);
        break;
    else
        set(handles.Recieve4,'String','No Data Recieved, Check your Connection')
        if m==len
             break;
        end
    end
    end
end
if f==0
    m=0;
end
flushinput(w);
flushoutput(w);
try
     fclose(w);
 end
```

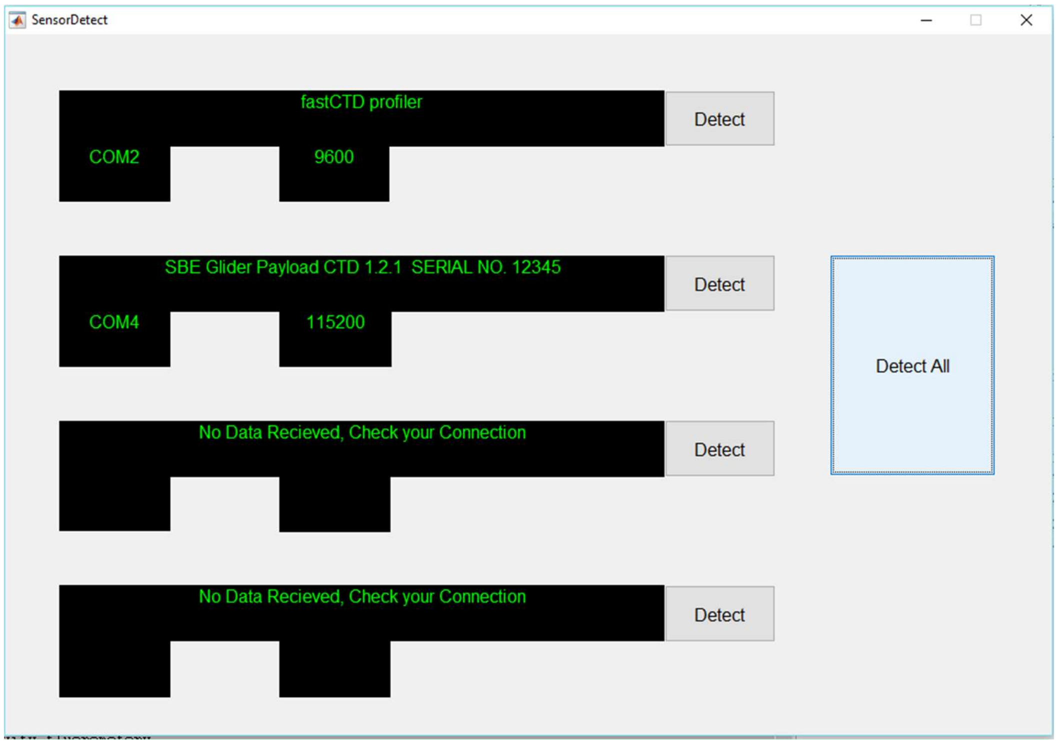
```
% --- Executes on button press in DetectAll.
function DetectAll_Callback(hObject, eventdata, handles)
% hObject    handle to DetectAll (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
global outa outa1 outa2 outb outb1 outb2 outc outc1 outc2 outd outd1 outd2 comA bA
comB bB comC bC comD bD;
Detect1_Callback(handles.Detect1, eventdata,handles);
Detect2_Callback(handles.Detect2, eventdata,handles);
Detect3_Callback(handles.Detect3, eventdata,handles);
Detect4_Callback(handles.Detect4, eventdata,handles);
s=serial(comA);
set(s,'BaudRate',bA,'Terminator','CR');
fopen(s);
 while(1)
     flushinput(s);
     outa2=fscanf(s);
     if ~strcmp(outa1, outa2) && ~strcmp(outa, outa2)
         fi1=fopen('Data.txt','a');
         fprintf(fi1,'%s %s', 'Sensor1 ');
         fprintf(fi1,'%s %s\r\n', outa2);
         fprintf(s,'data recieved');
         break;
     end
 end
 fclose(fi1);
 try
     fclose(s);
 end
 d=serial(comB);
set(d,'BaudRate',bB,'Terminator','CR');
fopen(d);
  while(1)
     flushinput(d);
     outb2=fscanf(d);
     if ~strcmp(outb1, outb2) && ~strcmp(outb, outb2)
         fi2=fopen('Data.txt','a');
         fprintf(fi2,'%s %s', 'Sensor2 ');
         fprintf(fi2,'%s %s\r\n', outb2);
         fprintf(d,'data recieved');
         break;
     end
 end
 fclose(fi2);
 try
     fclose(d);
 end
  a=serial(comC);
set(a,'BaudRate',bC,'Terminator','CR');
fopen(a);
  while(1)
     flushinput(a);
     outc2=fscanf(a);
     if ~strcmp(outc1, outc2) && ~strcmp(outc, outc2)
         fi3=fopen('Data.txt','a');
         fprintf(fi3,'%s %s', 'Sensor3 ');
         fprintf(fi3,'%s %s\r\n', outc2);
         fprintf(a,'data recieved');
         break;
     end
 end
 fclose(fi3);
 try
     fclose(a);
 end
 w=serial(comD);
set(w,'BaudRate',bD,'Terminator','CR');
fopen(w);
  while(1)
     flushinput(w);
```

```
        outd2=fscanf(w);
        if ~strcmp(outd1, outd2) && ~strcmp(outd, outd2)
            fi4=fopen('Data.txt','a');
            fprintf(fi4,'%s %s', 'Sensor4 ');
            fprintf(fi4,'%s %s\r\n', outd2);
            fprintf(w,'data recieved');
            break;
        end
    end
    fclose(fi4);
    try
        fclose(w);
    end
```

8. Output

The output of the program comprises of:

- The GUI Displaying the Information of the sensor connected, the COM port at which it is connected and the baud rate used for the serial communication. Here two sensors instruments are connected- The fastCTD profiler by Valeport in COM2 at 9600 baud rate and The Glider Payload CTD by Seabird Scientific in COM4 at 115200 baud rate (All these simulated using the terminal)



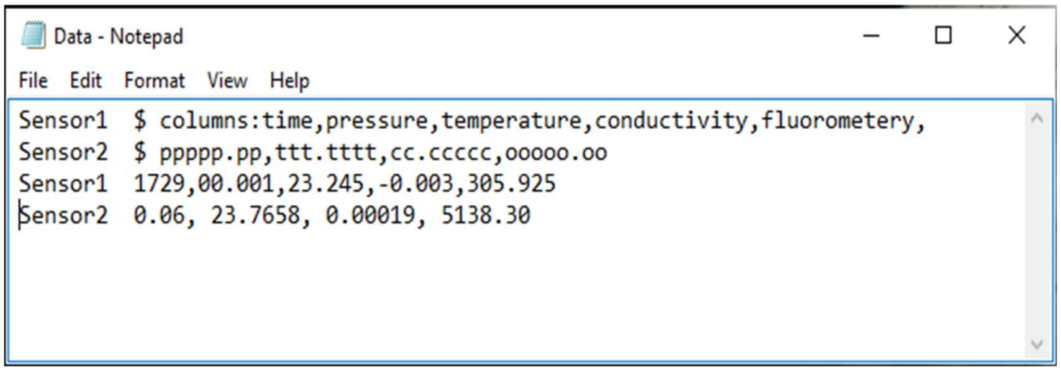
fastCTD profiler
COM2 9600

SBE Glider Payload CTD 1.2.1 SERIAL NO. 12345
COM4 115200

No Data Recieved, Check your Connection

No Data Recieved, Check your Connection

- A text file “Data.txt” being created with the format of the output readings of each sensor followed by their readings



```
Sensor1  $ columns:time,pressure,temperature,conductivity,fluorometery,
Sensor2  $ ppppp.pp,ttt.tttt,cc.ccccc,ooooo.oo
Sensor1  1729,00.001,23.245,-0.003,305.925
Sensor2  0.06, 23.7658, 0.00019, 5138.30
```

9. Conclusion

Market is flooded with different make and brands of sensors with different levels of sensitivity and endurance. These Sensors are used based on the requirement and fetching data from each of sensor requires different software interface. Here, in this project an effort has been made to develop a common software interface that could fetch data from different brands of sensors which requires detection of output parameters of sensors such as baud rate and format for the output. Using MATLAB and other simulation tools like Terminal and Virtual Serial Port Driver, it has been successfully demonstrated that a universal code can be used as interface for fetching data from different kinds of sensors. Here the program can fetch data from up to four sensors simultaneously which can be enhanced based on requirement.

10. References

Baker D. J. 1981 "Ocean instruments and experiment design" Chapter 14 pp 416–418, available at http://ocw.mit.edu/resources/res-12-000-evolution-of-physical-oceanography-spring-2007/part-3/wunsch_chapter14.pdf (https://en.wikipedia.org/wiki/CTD_(instrument))

CTD Instrument and Water Sampler". Alfred-Wegener-Institute for Polar- and Marine Research. Retrieved 19 September 2012. (https://en.wikipedia.org/wiki/CTD_(instrument))

Pickard George L. and William J. Emery "Descriptive Physical Oceanography, An introduction" 5th ed., Butterworth-Heineman (Elsevier Science): 1990 (https://en.wikipedia.org/wiki/CTD_(instrument)

Webopedia (2003-09-03). "What is serial port? - A Word Definition From the Webopedia Computer Dictionary". Webopedia.com. *Retrieved 2009-08-07*. (https://en.wikipedia.org/wiki/Serial_port)

Axelson, Jan (1 September 2006). *USB Mass Storage: Designing and Programming Devices and Embedded Hosts* (1st ed.). *Lakeview Research*. *ISBN 978-1-931-44804-8*. (https://en.wikipedia.org/wiki/USB)

——— (1 December 2007). *Serial Port Complete: COM Ports, USB Virtual COM Ports, and Ports for Embedded Systems* (2nd ed.). Lakeview Research. *ISBN 978-1-931-44806-2*. (https://en.wikipedia.org/wiki/USB)

Ralph Santitoro (2003). "Metro Ethernet Services – A Technical Overview" (PDF). mef.net. Retrieved 2016-01-09. (https://en.wikipedia.org/wiki/Ethernet)

11. Figures

12. Glossary

1.	ADCP	Acoustic Doppler Current Profiler
2.	AHRS	Attitude and Heading Reference System
3.	BT	Bathythermograph
4.	CTD	Conductivity, Pressure And Depth
5.	DVL	Doppler Velocity Log
6.	IMU	Inertial Measurement Unit
7.	USB	Universal Serial Bus
8.	VSPD	Virtual Serial Port Driver
9.	MATLAB	Matrix Laboratory
10.	GUI	Graphical User Interface