2015 International Nuclear Atlantic Conference - INAC 2015
São Paulo, SP, Brazil, October 4-9, 2015
ASSOCIAÇÃO BRASILEIRA DE ENERGIA NUCLEAR - ABEN
ISBN: **978-85-99141-06-9**

PROPOSAL OF A SYNCHRO PANEL METER INSTRUMENT TO REPLACE THE OBSOLETE SYNCHRO/RESOLVER READING DEVICE USED AS POSITION INDICATOR OF SAFETY RODS ASSEMBLY OF THE BRAZILIAN IEA-R1 NUCLEAR RESEARCH REACTOR

Fabio de Toledo*, Franco Brancaccio and José Patricio N. Cárdenas

Instituto de Pesquisas Energéticas e Nucleares (IPEN / CNEN - SP)
Av. Professor Lineu Prestes 2242
05508-000 São Paulo, SP
fatoledo@ipen.br
fbrancac@ipen.br
ahiru@ipen.br

ABSTRACT

IPEN (Instituto de Pesquisas Energéticas e Nucleares) was founded in 1956 (as Atomic Energy Institute – IEA) as a facility complex, for the research, development and application, in the nuclear technology field. The institute is recognized as a national leader in nuclear research and development (R&D), including the areas of reactor operation, radiopharmaceuticals, industrial and laboratory applications, materials science and laser technologies and applications. IPEN's main facility is the IEA-R1, nuclear research reactor (NRR), today, the only one in Brazil with a power level suitable for applications in physics, chemistry, biology and engineering. Some radioisotopes are also produced in IEA-R1, for medical and other applications. A common problem faced in the IEA-R1mantainenence is instrumentation obsolescence; spare parts are no more available, because of discontinued production, and an updating program is mandatory, aiming at modernization of old-aged I&C systems. In the presented context, an electronic system is here proposed, as a replacement for the reactor safety (shim) rods assembly position indicator, based on an open-source physical computing platform called Arduino, which includes a simple microcontroller board and a software-code development environment. A mathematical algorithm for the synchro-motor signal processing was developed, and the obtained resolution was better than 1.5%.

1. INTRODUCTION

IEA-R1 is a swimming pool type reactor, moderated and cooled by light water, using graphite and beryllium as reflectors. First criticality was achieved on September 16th, 1957. IEA-R1, currently, operates at 4.5 MW, with an operational schedule of continuous 64 hours a week.

The main purposes of IEA-R1 NRR are: production of radioisotopes; fuel and structural materials testing, for nuclear power engineering; neutron radiography; neutron activation analysis; neutron transmutation doping; research in neutron and condensed matter physics.

One of the most important components of the reactor is the safety and control rod assembly, used in the reactivity rods calibration, core configuration changes and during reactor starting, criticality and steady state operation. Such assembly is monitored by a rod position indicator instrument, generally a rotating device which provides suitable output signal(s) that can be converted into the device's shaft-angle, by means of a proper decoding circuitry.

There are different rotary transducers, such as [1]: potentiometers; brush encoders; optical shaft encoders; synchro/resolvers, Rotary Variable Differential Transformers (RVDTs) and Linear Variable Differential Transformers (LVDTs).

Differently from potentiometers and brush transducers, moving electrical contacts are absent in Synchro/Resolver, RVDT and LVDT devices, which exhibit good performance, almost unaffected by aging or wear and temperature changes [1]. Such characteristics make them suitable for reliable applications, such as for nuclear reactor power controlling and monitoring, and servo applications which require absolute mechanical position sensors.

The Synchro/Resolver category (synchro, for short), subject of this work, is basically constituted by rotating transformers, where the sinusoidal input reference, U_0, is amplitude modulated, into output signals. For instance, Fig. 1 [2] shows a synchro device which modulates the input reference, U_0, with trigonometric values of the shaft angle, ε, into two output signals, U_1 (sine modulation) and U_2 (cosine modulation).

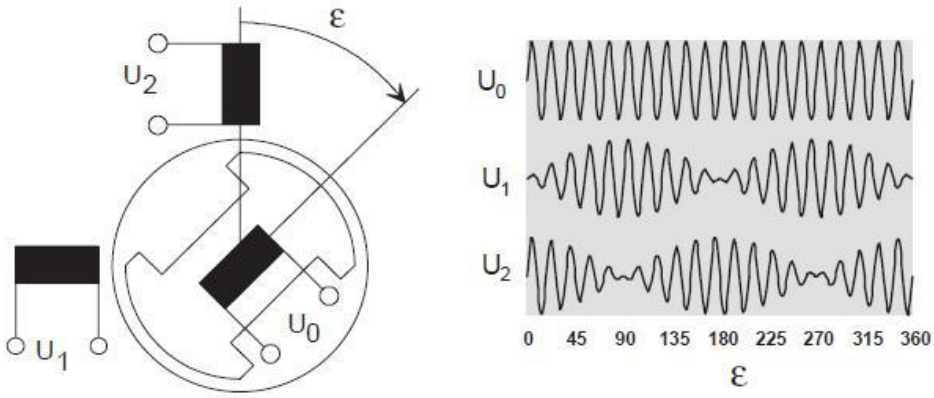


Figure 1: Synchro/Resolver simplified functional diagram and corresponding signals [2]: input reference, U_0; output sine, U_1 and cosine, U_2; ε represents the shaft angle.

The IEA-R1 safety and control rod assembly is provided with a position indicator called Synchro Panel Meter (SPM), composed of a synchro/resolver and digital converter, including custom logic devices, sophisticated transformers, solid state A/D and error processing circuits, in order to insure the analog-to-digital conversion (1975 digital technology). The converted synchro/resolver data is a three-digit integer, ranging from 000 to 999, visualized through Beckman raised-cathode, planar-gas-discharge 7-segments display, and representing a rod displacement from 0 to 60 centimeter. SPM electronics is now affected by aging, and corrective actions are becoming more frequent and difficult, due to unavailability of commercial spare parts. In order to overcome these difficulties, a new electronic readout system was proposed, based on microcontroller techniques.

2. GENERAL DESCRIPTION OF THE PROPOSED SYNCHRO READOUT

Present section shows the approaches adopted in circuit implementation, focusing on the readout circuitry development and testing. Therefore, the synchro signals conditioning and generation of DC readable levels are planned as the development's next-step.

2.1. Implementation Platform

Proposed hardware is based on the ARDUINO open platform [3] (including hardware and software), created in 2005, to provide a powerful and easy-to-use development tool-set. Different microcontroller boards are available, such as UNO, MEGA, LEONARDO and GALILEO, suitable for the implementation of interactive systems (controlling, monitoring, robotics, etc.). Program coding and uploading are accomplished by an Integrated Development Environment (IDE) which provides C/C++ compatibility, including the AVR Libc [4]. Free software includes libraries written for a wide range of peripheral devices, such as LED's and displays, stepper and servo motors, relays, sensors, Ethernet boards and others. Yet simple, UNO board, shown in Fig. 2, was chosen for development. Equipped with the Atmel AVR microcontroller ATmega328, UNO fulfils all the project requirements.

Figure 2: ARDUINO UNO board [3], employed for development of the readout module.

ARDUINO UNO main specifications and features are [3]:

Microcontroller	ATmega328
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limits)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
Analog Input Pins	6 (10 bits Analog to Digital Conversion resolution)
DC Current per I/O Pin	40 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g

2.2. Synchro Shaft Encoder

Since the readout circuit represents the main target at this stage, synchro signals conditioning is neglected and readable DC voltage levels are derived directly from the reactor's instrument, SR-300 Shaft Encoder/Angle indicator [5], as shown in Fig. 3.

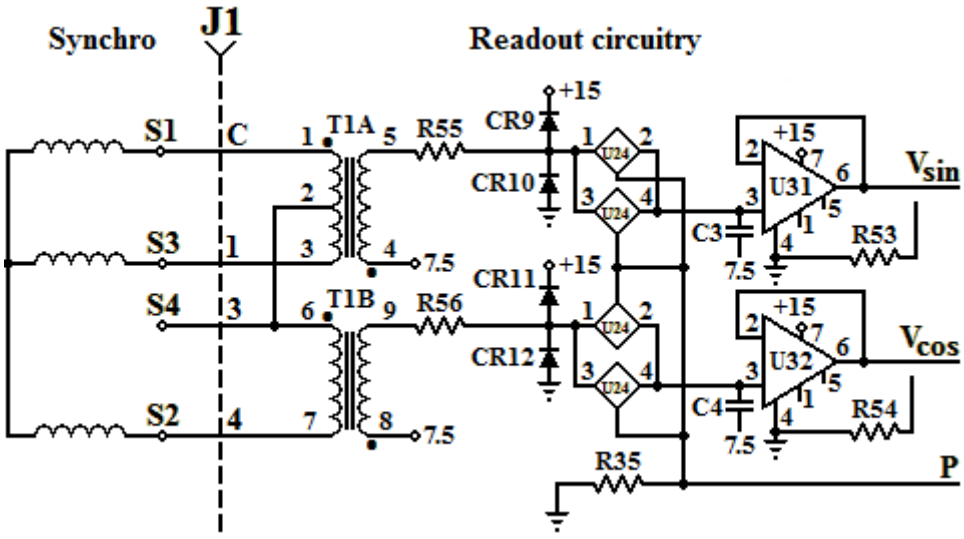


Figure 3: Signal conditioning stage diagram of SR-300 Shaft Encoder/Angle Indicator [5]. DC voltage levels V_{sin} and V_{cos} are obtained from the Synchro AC voltages, corresponding to sine and cosine signals.

2.3. Proposed Synchro Readout Circuitry

The proposed circuit is very simple, as seen in Fig. 4, since the most of the tasks is performed by software (from now on, ARDUINO UNO board will be referred as ARDUINO).

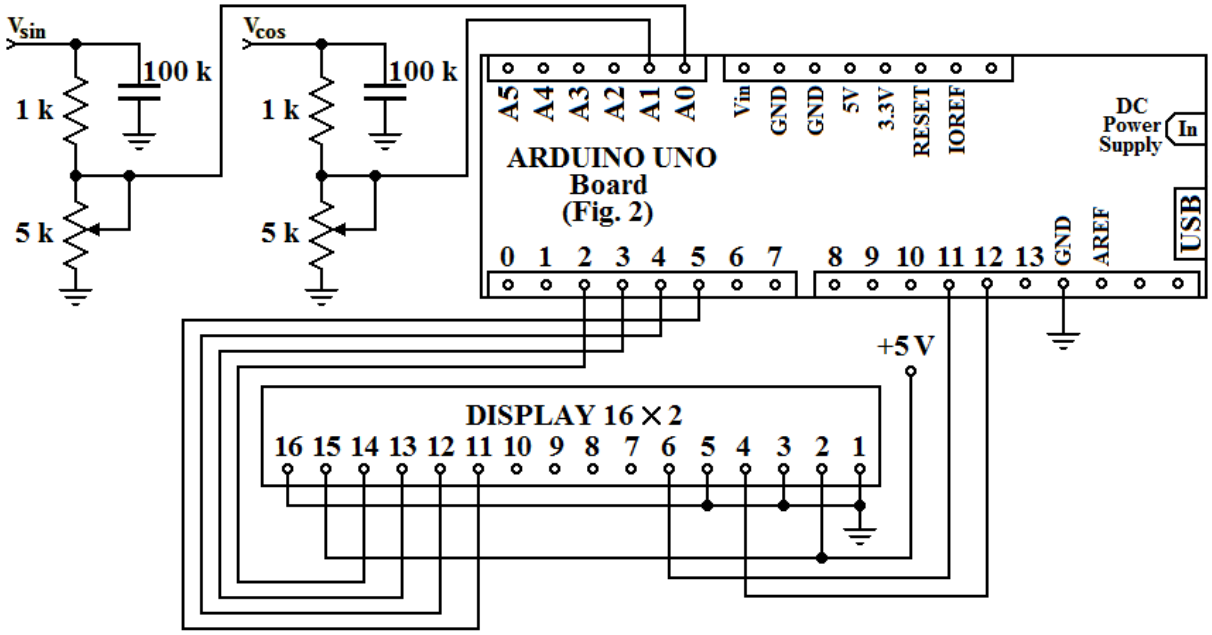


Figure 4: Diagram of the proposed readout circuit, based on the ARDUINO board.

DC voltages from the Shaft Encoder (V_{sin} and V_{cos}, Fig. 3) are connected to voltage dividers and then to ARDUINO analog inputs, A0 and A1. Shaft Encoder voltages range (approximately, from 4.4 to 11.6 V) is incompatible with ARDUINO input specifications (from 0 to 5 V) and, then, voltage dividers were provided for range trimming. Actually, with such a simple approach, the readable values remain within, approximately, from 2 to 5 Volts. Such voltages (A0 and A1 analog inputs) are read inside the program's loop, converted into the rod position (0 to 999) and shown in a two-line-sixteen-column liquid crystal display (32 characters).

2.4. ARDUINO Software Coding – Sketch

ARDUINO programs are made up of two main C/C++ type functions, setup and loop, with the following prototypes: void setup(); void loop().

As seen, both functions have no return (void) and receive no arguments (empty parameter list). The setup function performs some desired tasks once, at system starting (devices configurations and others). The loop function is continuously executed (loop-calling), as denoted by its name.

An ARDUINO coding-project is called a Sketch. Environment provides a template at a new-Sketch opening, containing the two above mentioned functions, as shown in Table 1.

Table 1: Empty Sketch template provided by ARDUINO coding environment

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Setup and loop functions, initially empty, are implemented by the programmer to perform the proper tasks. For the presented readout system, setup and loop functions perform, respectively: the display configuration; the reactor's rod position reading and displaying.

Extra code can be inserted before and after the ARDUINO main functions. Before functions, header files can be included, global variables can be defined/initialized, user functions can be implemented or prototyped. If user functions are just prototyped, their implementations will come after ARDUINO main functions. Programming can assume, either pure C or C++ formats, and user files can be written and added to the Sketch and even added as user defined libraries, to be included to any Sketch. Then, user functions, classes and any other C/C++ stuff are allowed to be created and added. As usual, the header/implementation format should be used. For instance, a class or set of related classes can be 'declared' in a header file (*.h) and 'implemented' in a main file (*.cpp).

2.5. Readout Software

The readout main program code is quite simple: a few useful variables are defined at global scope; setup function configures the display; loop function reads the rod position, sending its value to the display device.

The most of program functionality and variables are defined by C++ classes, in the Object Oriented Programming (OOP) approach. Therefore, a unique object (instance) of the class Synchro, defined at global scope, encapsulates other objects (Composition/Aggregation) and the interface (member functions) which performs the required tasks.

2.5.1. C++ implemented classes

This subsection presents an overview of implemented classes, explaining their main functionalities provided by class' members (data and interface). Tasks are performed in steps, from DC levels reading (A0 and A1 analog inputs) to rod position displaying. Subsection lower levels try to follow an understanding sequence of classes, since some of them depend on the definition of others. Identifiers used in the code are bold faced.

2.5.1.1. Class Range

The class **Range** defines a contiguous interval of Real numbers within the **minimum** and the **maximum** values (inclusive), [**minimum**, **maximum**]. Real numbers are represented by double type variables (C/C++double precision floating point number). **Range**'s constructor overloading allows the instantiation of null ranges (default: [0.0, 0.0]) and defined ranges, [min, max], initialized from two double constants or from another previously defined **Range** instance (copy constructor). Limits of an already defined range can be read by 'get' and changed by 'set' member functions. **Range** instances are owned by instances of other classes.

2.5.1.2. Class StraightLine

StraightLine establishes a relationship between two **Range** instances, by obtaining the straight line equation which converts a value from an origin range into the value of a target range. The double type members, **intercept** and **slope** are computed from two given ranges: originRange = [minO, maxO]; targetRange = [minT, maxT]. Then, a value within originRange, oVal:double (abscissa), is converted into a value within targetRange, tVal:double (ordinate), by calculation: tVal = **slope** × oVal + **intercept**. Such operation is achieved by the member function **getOrdinate**, as follows: tVal = straightLineInstance.**getOrdinate**(oVal);.

StraightLine objects can be instantiated (constructor functions) or changed ('set' functions) by passing four double or two **Range** arguments.

Members **intercept** and **slope** also can be returned by 'get' member functions, **getSlope**(), **getIntercept**().

2.5.1.3. Class ArduinoConfig

ArduinoConfig provides two static (class scope) constant ranges which store the ARDUINO configuration. By Composition, the **Range** type members, **anaRange**, [0, 1023], and **voltRange**, [0.0, 5.0] V. Such ranges correspond, respectively, to the interval of analogic input read values and its equivalent voltage range. The class also provides static member functions to get the correct voltage from an 'int' given analog value, **ArduinoConfig::getVoltage**(anaVal), and to read an ARDUINO analog pin and return the voltage instead of an analog value, **ArduinoConfig::readVoltage**(pinNumber). Since all members are statically accessible, no instantiation is needed (but possible).

2.5.1.4. Class Angle

An **Angle** instance represents an arc used for trigonometric calculation. Arc value can be defined at construction time (0.0 as default), read by 'get' function and redefined, at any time, by 'set' function. RADIAN and DEGREE unities, defined as enumeration constants (enum **AngleUnit**), can be passed as arguments to function members (RADIAN is default), in order to get or set angles with the desired or appropriate unit (radian or degree).

Class **Angle** also provides statically accessible (class scope) **Range** instances, by Composition, defining:

◦ trigonometric sine and cosine values: **trigoRange** = [-1.0, 1.0];
◦ one revolution angle values (radian): **angleRangeRad** = [0.0, 2.0π);
◦ one revolution angle values (degree): **angleRangeDeg** = [0.0, 360.0);

Such internal static ranges are useful for the proper conversion from read voltages to rod position, as explained in next subsection.

2.5.1.5. Class Synchro

Class **Synchro** includes all the above functionality, by means of associations (Aggregation, Composition or use) with the other classes (subsections 2.5.1.1. to 2.5.1.4): by Composition, **Angle** and **Range** objects are owned by a **Synchro** instance which also uses functions implemented in **StraightLine** and **ArduinoConfig**. A unique instance of class **Synchro** is required to perform the entire task, put inside the loop function, as mentioned in 2.5 (ARDUINO readout Sketch).

For a better understanding, fragments of code (Sketch files) are reproduced in Table 2 and Table 3. Table 2 highlights the relevant statements from the Sketch's main file (synchroCode.ino). Table 3 shows the **Synchro**'s 'private' data members (from class declaration file: synchro.h). Based on such tables, the sequence of the program's steps is descripted.

Table 2: Fragment of code, synchroCode.ino file, showing relevant statements. Only two devices are required for the main function implementation ('loop'): Display and Synchro, internally represented by their respective global instances: 'lcd' and 'synchro'

```
// here, the ARDUINO's defined headers (used in Sketch)
#include <Arduino.h>                    //: definition of ARDUINO's stuff
#include <LiquidCrystal.h>              // definition of class LiquidCrystal

LiquidCrystal lcd(12, 11, 5, 4, 3, 2);  // construction of the display object

// here, the headers defined in the 'synchroCode' Sketch
#include "angle.h"              // definition of class Angle
#include "range.h"              // definition of class Range
#include "straightline.h"       // definition of class Range
#include "arduinoConfig.h"      // definition of class ArduinoConfig
#include "synchro.h"            // definition of class Synchro

// Global variables
// Synchro setting constants
const double
        VOLTAGE_MIN     = 2.0,          // Synchro's minimum output voltage
        VOLTAGE_MAX     = 5.0,          // Synchro's maximum output voltage
        ROD_MIN         = 0.0,          // Reactor's rod-position minimum-reading (0.0 rad)
        ROD_MAX         = 1000.0;       // Reactor's rod-position maximum-reading (2pi rad)

// synchro object construction (also called instantiation; refer to Table 3 and further explanation)
Synchro
        synchro(VOLTAGE_MIN, VOLTAGE_MAX, ROD_MIN, ROD_MAX, A0, A1);
// end of Global variables

void setup() {
        lcd.begin(16, 2);               // configuration of display device
        lcd.print("Synchro/Resolver");  // sending message to display device (Title)
}

void loop() {
        // Synchro reading and calculation: reads voltages and calculates the rod position
        synchro.readRodPosition();

        // displaying angle (degree): sets display cursor position and prints shaft angle value
        lcd.setCursor(1, 1);
        lcd.print(synchro.getAngle(DEGREE));

        // sets display cursor position for rod position displaying
        lcd.setCursor(9, 1);

        . . . // here, some setting statements for display formatting

        // displaying rod position (synchro's member, since it is obtained from synchro's angle)
        lcd.print(synchro.getRodPosition());

        delay(200);     // waits 200 ms before next loop calling (rod position updating)
}
```

Table 3: Fragment of class Synchro definition code, header file synchro.h, showing the relevant data members, with 'private' visibility

```
Range
        outVoltRange,       // Synchro output voltage range, [2.0, 5.0] V
        rodPosRange;        // Synchro shaft range position, [0, 1000)

int
        v1,                 // Analog A0 input read value for sine, [0, 1023]
        v2,                 // idem for cosine (A1 input)
        v1Pin,              // ARDUINO pin-number for v1 analog-input (pin A0)
        v2Pin,              // idem for v2 (A1)
        rodPos;             // Reactor's rod position (obtained from shaft angle)

Angle
        angle;              // Synchro's shaft angular-position, [0, 360) (degree)

double
        voltage1,           // voltage value for sine, obtained from v1
        voltage2,           // idem, for cosine, obtained from v2
        sineValue,          // shaft angle's sine-value, obtained from voltage1
        cosineValue;        // shaft angle's cosine-value, obtained from voltage2
```

The **Synchro**'s constructor version presented in Table 2 is prototyped as follows: **Synchro**(**double, double, double, double, int, int**); the six received arguments are:

◦ the first two 'double' define the range of read voltages, at ARDUINO pins **A0** and **A1**:
outVoltRange member initialization: [2.0, 5.0] (values can be trimmed);

◦ the last two 'double' define the integer range of reactor's rod position:
rodPosRange member initialization: [0, 999] or [0, 1000) (integer);

◦ the two 'int' inform to the Synchro object the analog pin numbers to be read (**A0** and **A1**):
v1Pin and **v2Pin** members' initialization: ARDUINO defined constants, **A0** and **A1**.

Rod position is obtained by the function call, **synchro.readRodPosition**(); other member functions are internally called in order to set **angle** and **rodPos** members' values, following sequence of tasks enumerated below:

◦ **voltage1** and **voltage2** are calculated from reading **A0** and **A1** analog pins, into **v1** and **v2**:
v1 and **v2** values remain within **Arduino::anaRange**, [0, 1023];
voltage1 and **voltage2** values remain within **Arduino::voltRange**, [0.0, 5.0];
Note: actually, values are constrained by the voltage dividers (Fig. 4) to [2.0, 5.0 V];

◦ **voltage1** and **voltage2** are converted, respectively, into **sineValue** and **cosineValue**:
values remain within **Angle::trigoRange,** [-1.0, 1.0];

◦ **angle** is then obtained from **sineValue** and **cosineValue** by cmath atan2 function,
converting its original output range, [$-\pi$, π] (radian) to, either
Angle::angleRangeRad, [0.0, 2.0π) (radian) or
Angle::angleRangeDeg, [0.0, 360.0) (degree);

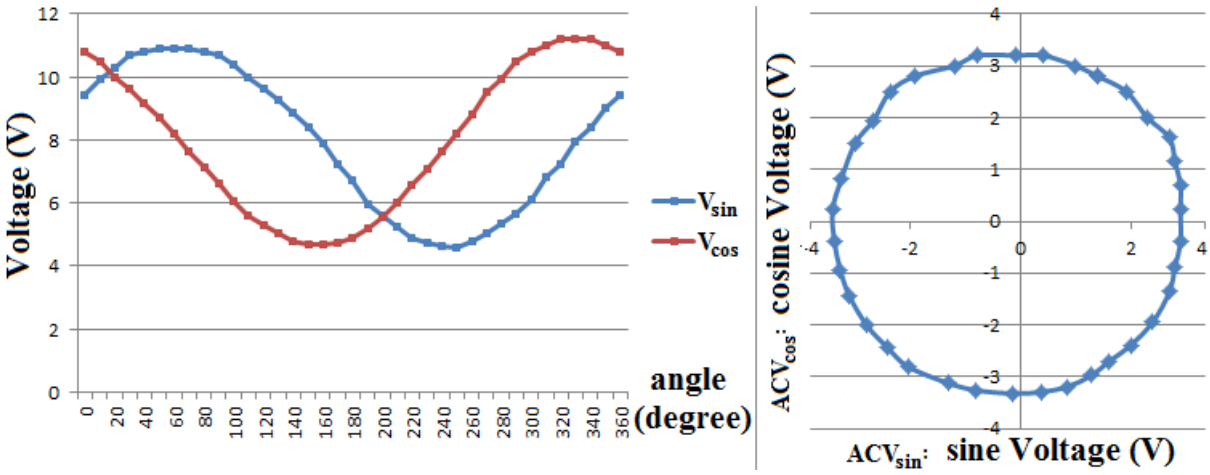
◦ **angle** is finally converted into the **rodPos** (an integer value which is then displayed):
values remain within **Synchro::rodPosRange,** [0, 999]; Note: 1000 is converted to 0.

3. EXPERIMENTAL RESULTS

One of the SR-300 Shaft Encoder/Angle indicator units (2.2.) was put available for experimentation in the electronics development laboratory. V_{sin} and V_{cos} outputs were derived, connected to new readout circuitry (Fig. 4) and measured, varying the shaft angle value from 0 to 360 with 10 degree steps (approximately).

3.1. Shaft Encoder Voltages

Fig. 5 (a), below, plots both voltages as function of angle (degree). Despite of a phase shift, sine and cosine behavior can be denoted (V_{sin}: sine; V_{cos}: cosine). Fig. 5 (b) plots the AC component of V_{cos} as function of the AC component of V_{sin}, showing the composed circumference.



(a) Shaft Encoder voltages versus angle **(b) sine versus cosine voltage**

Figure 5: SR-300 Shaft Encoder/Angle indicator voltages:
(a) V_{sin} and V_{cos} as function of the shaft angle;
(b) cosine as function of sine (AC voltage components).

Starting shaft position previous setting was not performed leading to the phase shift shown in Fig. 5 (a). Such setting can be accomplished, both, by the proper shaft positioning (mechanically) or, if desired, by automatic setting of the angle phase variable (software).

3.2. Readout

Since the SR-300 Shaft Encoder/Angle indicator was employed to supply the conditioned DC levels (V_{sin}, V_{cos}), the readout of such module could be directly compared to that obtained with the proposed circuit (which also provides the angle value in degrees).

Comparison was obtained by plotting together the two readout values, varying the shaft angle from 0 to 360 with 5 degrees steps, as seen in Fig. 6.

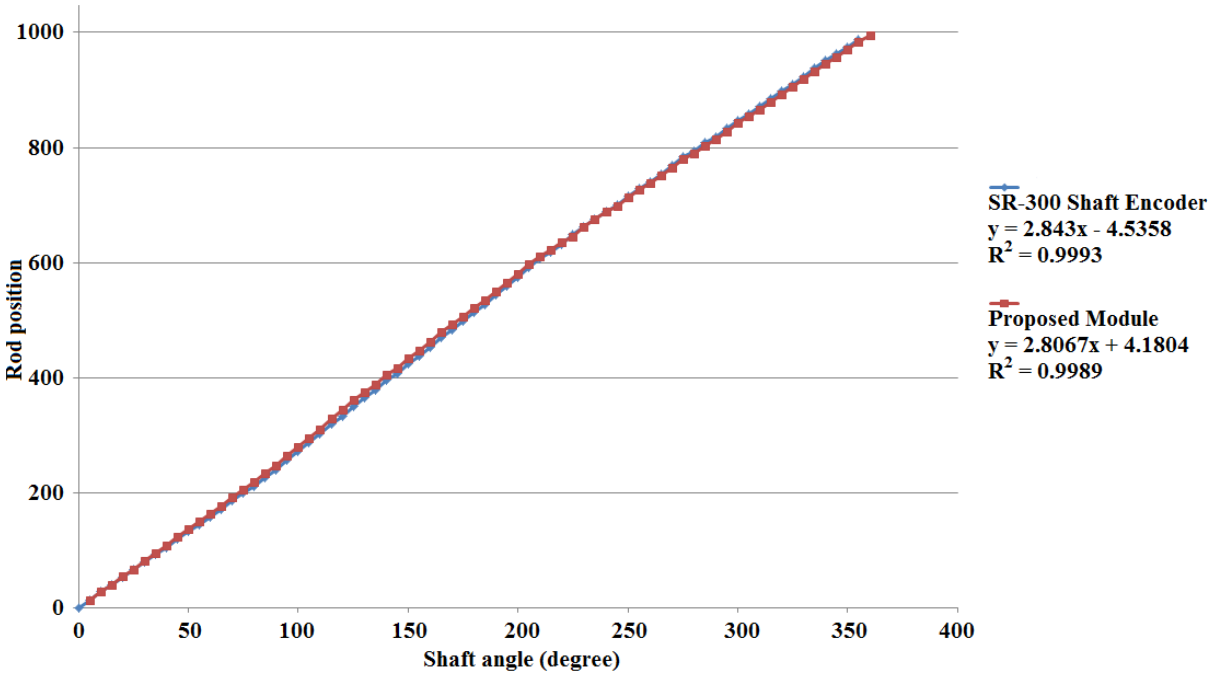


Figure 6: Plotting of SR-300 Shaft Encoder/Angle readout (blue points) compared to the proposed circuit values. Angle was varied with 5 degrees steps.

Readout range or rod position, [0, 999], corresponds to the original SR-300 Shaft Encoder range, representing a rod displacement from 0 to 60 centimeter. Difference between readout values is 1.3%, as shown by the tendency lines fitting, obtained with Excel program. Probably, such difference could be reduced with angle-phase trimming (software). A small non-linearity is also observed in the proposed circuit plot (red line in Fig. 6), probably caused by voltage reading fluctuations, non-linearity in analog to digital conversion or by a slight difference between the sine and cosine voltage ranges. Although such discrepancies can be considered irrelevant, improvements can be done either in the circuit or in the software. For instance, readout linearity will improve if fitted-values are displayed instead of direct calculation values.

4. CONCLUSIONS

The proposed Readout Circuitry showed satisfactory results, with a difference smaller than 1.5% to usual module. Circuit and software improvements are planned, as well as the development of the circuit for the Synchro's AC signals conditioning. Therefore, the new unit will constitute a direct replacement to the old modules, depending on just the available Synchro devices.

ACKNOWLEDGMENTS

Authors acknowledge Mr. Algeny Vieira Leite who provided the SR-300 Shaft Encoder/ Angle module for experimentation and development of the new readout unit.

REFERENCES

1. "Synchro/Resolver Conversion Handbook", Fourth Edition, electronic version, 105 Wilbur Place, Bohemia, New York 11716-2482, http://www.ddc-web.com (2015).
2. Texas Instruments, Application Report, SPRA605 - February 2000.
3. Arduino, open-source electronics platform based on easy-to-use hardware and software, https://www.arduino.cc/ (2015).
4. AVR Libc free Software project providing high quality C library for use with GCC on Atmel AVR microcontrollers, http://www.nongnu.org/avr-libc/ (2015).
5. ILC Data Device Corporation, Bohemia, New York.