Master Thesis 2016

Candidate: Achema Hosea Egbubu (142773)

Title: Monitoring 50/60Hz Grid Coupling – A Study In Conjunction with Wind-Energy Feed to Main Grids

Telemark University College
Faculty of Technology
M.Sc. Programme

University of Applied Sc.
Faculty Technology, Wihelmshaven
Master Thesis

MASTER'S THESIS, COURSE CODE FMH606

Student: Achema Hosea Egbunu

Thesis title: Monitoring 50/60Hz Grid Coupling – A Study in Conjunction With Wind Energy Feed to Main Grids

Signature: . . . . . .

Number of pages: <#>

Keywords: . . . . . .

. . . . . .

. . . . . .

Supervisor:	Helge Lorenzen	Sign.: . . . . .
2nd Supervisor:	Prof. Josef Timmerberg	Sign.: . . . . .
3rd Supervisor:	Saba Mylvaganam	Sign.: . . . . .
Censor:	<name>	Sign.: . . . . .
External partner:	Wobben Research & Dev.	Sign.: . . . . .
Availability:	<Open/Secret>	
Archive approval (supervisor signature)**:**		Sign.: . . . . .
		Date : . . . . .

Abstract:

This master thesis is about investigating and comparing findings for different voltage control strategies for the 60Hz grid connection using microcontroller (Arduino). The P-Control, PI-Control, PID control and Open loop control were designed, implemented and their behavior based on setpoint jumps, active power jumps and reactive power jumps, etc were investigated and compared. Different hardware for measurements and varying the excitation current for the 60Hz machine were designed, built, tested and implemented for this project.

A comprehensive interface for monitoring and control (Man Machine Interface) for the user was also developed and implemented using web server. This interface is a powerful tool providing advance access to different control strategies, online hardware calibration, data logging and debugging tool.

Different software was also used during the project for simulation and data analysis.

Telemark University College accepts no responsibility for results and conclusions presented in this report.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This master thesis was created in collaboration between the University of Southeast Norway, the Jade University of Applied Sciences and its industrial cooperation partner Wobben Research and Development, which is a company within Germany's greatest group of Wind Energy-Converter-Manufacturer. This thesis is based on small scale energy systems operated for research purpose. The system is to be upgraded to improve possibilities for experiments especially with 60Hz grids. To get the 60Hz frequency stable grid, two mechanically coupled synchronous machines are used. The task is to stabilize the voltage of the 60Hz grid connection by identifying measuarands involved, investigate the different characteristic values for variables of three phase power systems (e.g. rms values, space phasors, and fundamental amplitude estimation), use sensor data fusion (e.g. voltage or current sensors) to get higher level information like active and reactive power as an input for the control strategies, implement different control strategies (e.g. P, PI, PID and Fuzzy logic) to stabilize the voltage using sensor data, use microcontroller system (Arduino) and MATLAB for the investigations and systematic comparison of the findings for different data acquisition and control strategies.

1.1 PERCULIARITY OF THIS PROJECT

The master project is not isolated but is to be seen as part of a complex overall project. This leads to special constraints and boundary conditions:

- The timely completion had to be guaranteed because of firmly agreed use of the results.
- The application projects make very high demands in terms of functionality, feel and reliability.
- In addition to providing functionality it is an important goal of the project to prepare aspiring engineers and junior engineers to meet the specific needs in a research company. The project staff should be qualified diversified as far as possible within the project and gain experience in team work on major projects.

For these reasons, the procedure in project processing is as follows:

- Under the direction and responsibility of an experienced engineer, primary options were discussed to divide the overall project into subtasks.
- The project leader created a work frame and a professional proposed solution for each of the required tasks, at least as a fallback position.
- Each entrant was challenged to deal, at least fundamentally, with each part of Aspect
- Each entrant was challenged to pursue his own solutions for one or more subtasks.
- The resulting parallel partial solutions (hard- and software as well) were compared in the group, findings, pros and cons of different approaches were discussed. The most reliable parts were combined into an overall solution.
- For building hardware, the team was supported by technicians and skilled workers from the workshop.

1.2 CHALLENGE DESCRIPTION

- Enable testing of control strategies for wind energy converter in a 60Hz Grids
- Implement and Test different kinds of voltage control strategies.
- Open loop control with variable setpoint
- Closed loop control
- P, PI and PID with variable setpoint and online adjustable parameters
- Fuzzy-Control with replaceable characteristic
- Create all required documents
- Meet adadditional task specifications
    - Haptic requirements
    - Safety requirements

1.3 BRIEF LITERATURE REVIEW

Voltage control is the ability to maintain a constant, uniform voltage under normal operating conditions when subjected to any disturbance (Kundur 1994). It is an important aspect of power system for securing a reliable system operation (Mousavi 2011). Voltage control is an integral part of the power system stability. To maintain the output voltage of the 60Hz three phase synchronous machine, the excitation current of the machine is controlled. There are various causes of voltage instability for the three phase sysnchronous machine connected to the grid. The change in load connected to the grid (Machine output), the disturbances between the machine and the load, distance between the source (machine) and the load, low source voltage and an adverse load characteristics (Manohar 2012). The change in voltage is directly related to change in load (Manohar 2012). As the load increases the output voltage droped or decreases. Voltage control can also be considered interms of system faults (on grid) and loss of load (Manohar 2012). The ability of the control system to maintain a constant voltage under these conditions is considered as voltage control. The voltage control service is usually provided from the machines and constomers within the controlled areas (Mousavi 2011). According to (Mousavi 2011) synchronous machines are considered as one of the backbones for voltage control, easy to control and less expensive. In order to have the desired voltage control for the 60Hz machine there is need to compare different voltage strategies available that can be used on the machine to enable proper choice based on the findings.

2 ENVIRONMENT

The environment consists of:

- Mechanically coupled 50/60Hz three phase synchronous machines
- Small scale electric power grid connected to the output of the 60Hz machine
- The wind energy converters
- And the research lab

The overview of the process is shown in *Figure 2-1*.

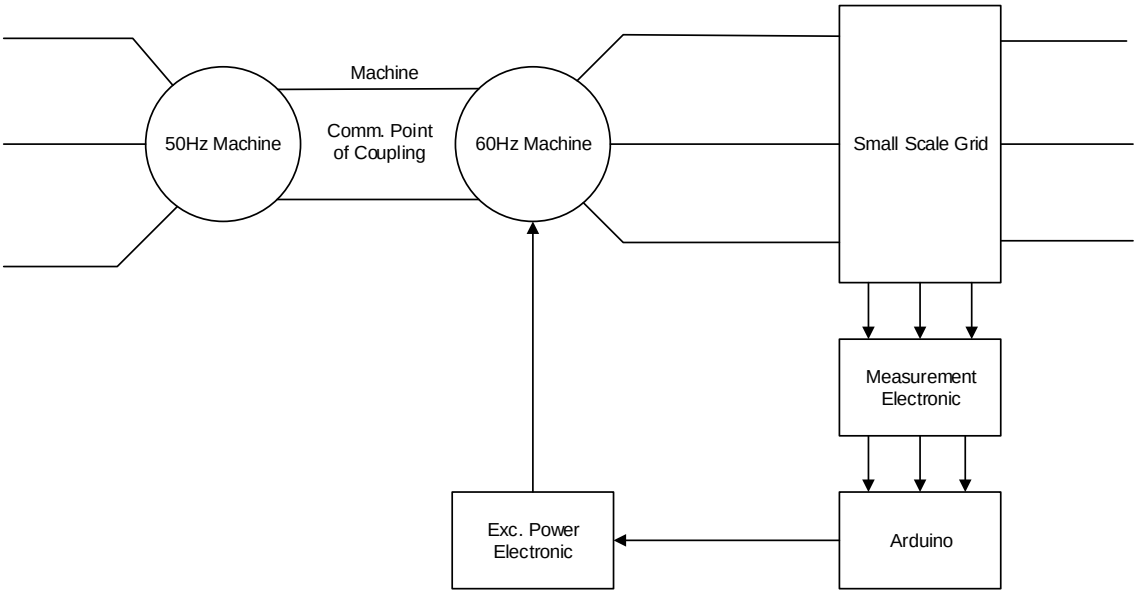


Figure 2-1: Overview of the Process.

2.1 SMALL SCALE ELECTRIC POWER GRID

Figure 2-2 shows the small scale electric power grid for the project. The output of the three phase 60Hz sysnchronous generator is feed to the grid.

Figure 2-2: Small Scale Power Grid Connected to the Output of the 60Hz Machine.

2.2 FREQUENCY TRANSFORMER UNIT

The whole machine system is referred to as the frequency transformer unit because it transforms 50Hz frequency to 60Hz and vise versa. The power can be made to flow in either direction. The set consists of two mechanically coupled synchronous machines and a power cabinet (Grid) in the basement as shown in *Figure 2-3*.

Figure 2-3: Mechanically Coupled 50/60Hz Machine (Frequency Transformer Units) Connected to the Power Grid.

As shown in the *Figure 2-3*, is a machine set of:

- Mechanically coupled synchronous machines. One machine is connected to the 50Hz, the other to the 60Hz grid.
- The frequency ratio is guaranteed by transmission (gear) ratio.
    - After a start sequence the 50Hz machine is directly connected to the lab-supply and has its own control for ramping up, which is out of scope in this project.
- The 60Hz machine is used as point of common coupling for the 60Hz experimental grid. The machine is generously sized to get in combination with a control of the excitation current a slack. The word “slack” is used in electrical power engineering for a voltage node, which voltage level may be viewed at as independent of load.
- The 60Hz machine has an additional winding as the source for the excitation current which is to be controlled.

Power Cabinet serves

- To receive all electronic and electromechanical components. It is connected via cable to the machine set and to other equipments in the laboratory through terminals.
- In addition it contains two outlets for data communication which leads to the IT-patch field of the lab.

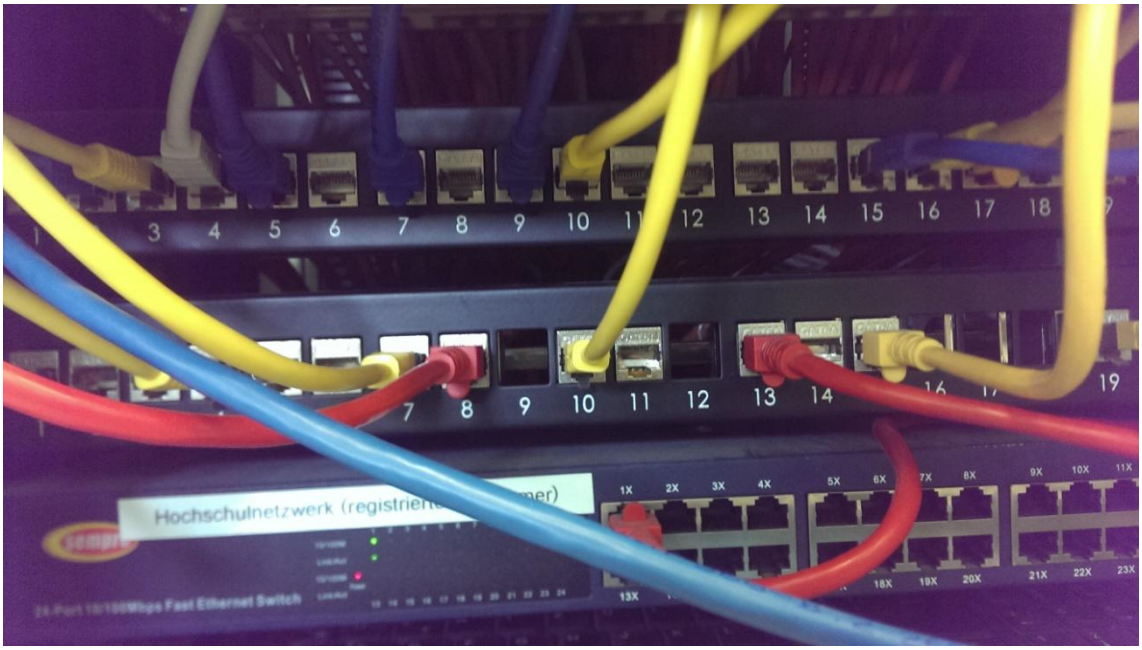


Figure 2-4: IT Patch Pannel where the Data Communication Ports from the Machine Power Cabinet (Power Grid) is Connected.

3 PRELIMINARY AND BASICS

This section describes the basic platform (Arduino), Arduino software architecture, the com port, data acquisition, application communication and mass storage of data used for the project implmentation.

3.1 PLATFORM USED (ARDUINO)

To the learning curve of the team members a development platform for the electronic control system known as Arduino was uniformly established. Arduino based on the definition from Arduino.cc,

"Arduino is an open-source prototyping platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing…." [https://www.arduino.cc/en/Guide/Introduction].

Throughout this thesis, it was hopeful clear that even very demanding projects can be realized with Arduino. Many things can be realized through simple calls of comfortable software libraries. But this is not enough when it comes to meet the higher demands and to be able to really exploit the potential of this platform.

Figure 3-1 shows the basic parts of Arduino board.

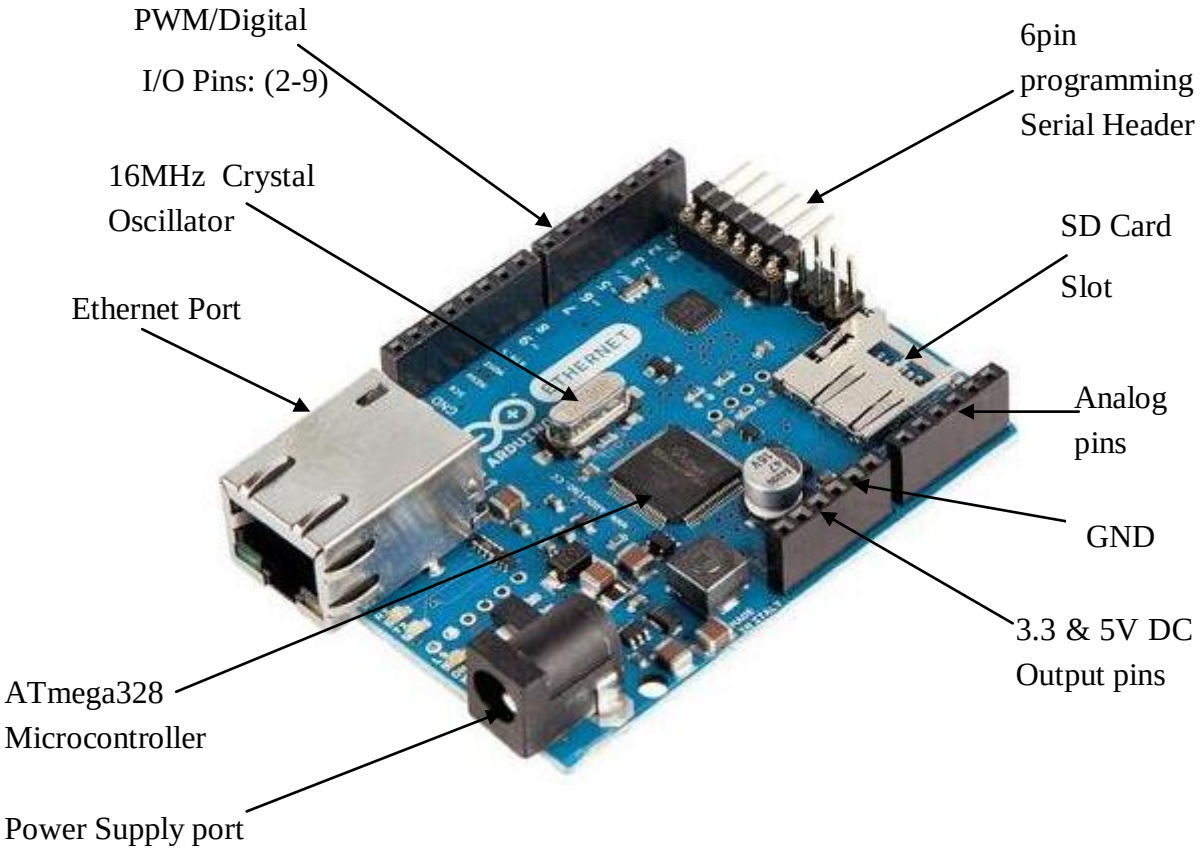


Figure 3-1: Arduino Ethernet Board (CONRAD 2016).

A lot of information for the hardware, IDE (Integrated Development Environment) and software libraries can be found starting at www.arduino.cc and in addition in many relevant forums.

The following description of ARDUINO features is limited to project-relevant aspects.

3.2 GENERAL ARDUINO SOFTWARE ARCHITECTURE

The basic structure of Arduino program is very simple. After switching on or reset, an initialization is first executed. Once this is completed, the loop function is called. This is unlimited repeated in accordance to the general concept. Custom Functions can be programmed and called up both in the setup-and the endless main loop function.

In addition, there is an option to assign functions to be called directly on hardware level when specific events occur. After finishing such a function, it is automatically returned to the position in main program where this was interrupted. This happens unnoticed

from the perspective of the main program. In this project are timer interrupt and an interrupt after completion of AD conversion which is of central importance.

This interrupt technique is a powerful tool from which this project extensive use is made. This allows the execution of commands, while passing time, which is needed by other parts of the controller or peripheral subsystems, for example in the conversion of analog values. The general Arduino software architecture is shown in *Figure 3-2*.

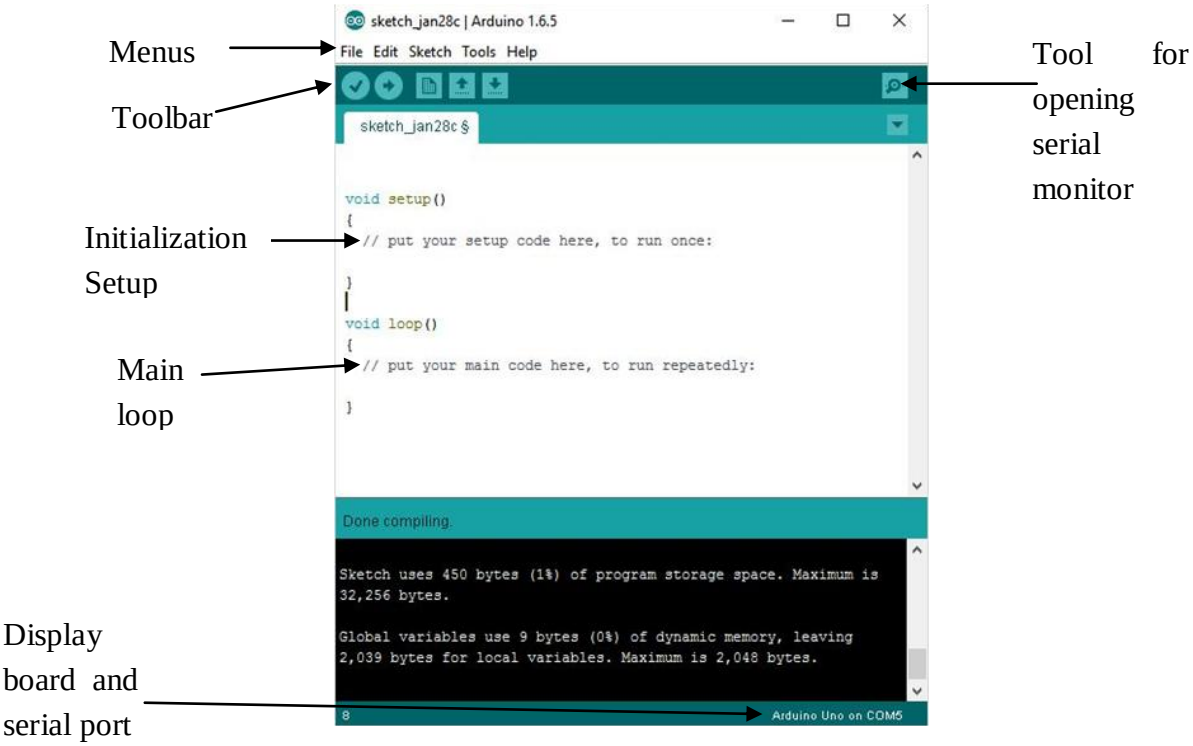


Figure 3-2: Arduino Integrated Development Environment (IDE) (Arduino 2016).

3.3 COM PORT

All Arduinos have at least one serial port. In this project it is used only for the transmission of the compiled program from the IDE controller and for simple issues in the first steps. For practical applications, the more powerful, but more complex to program Ethernet interface is used later.

3.4 DATA ACQUISITION INPUT

- In addition to the digital inputs, Arduino also has analog inputs. These are used in this project for getting an actual value.
- The actual AD conversion is done with a single controller internal converter.
- In order to realize a plurality of analog inputs this is preceded by a multiplexer. The conversion time is about 104us and can be reduced due to the registers of the controller in bit manipulations by direct limits; however, this has to be reckoned with by negative influence of resolution and / or accuracy.
- Another challenge arises when using the multiplexer. After changing the channel, a certain time is required to stabilize the voltage at the input of the converter at the level of the selected input channel.
- Select the channel, performing the conversion and reading of the sampled value is reduced to a single command when using the library function (*analogRead()*), but this simplification is to be paid with a high price: *Massive loss of performance!*

3.4.1 USE OF LIBRARY FUNCTION

- Syntax of the library function: *analogRead(pin).*

Action sequence of the library function:

- Set MUX to chosen analog input *pin*
- Start conversation
- Wait for the conversation to be done by the ADC-hardware. So the processor waits for about 104µs.
- Read conversation output registers to fetch data
- Combine bytes to a 16 bit integer result
- Put result to the stack and returned value to the function call.
- For the first read analog value after changing the input channel there is a big impact from the voltage level at the previous used input pin, because conversation is started immediately after the multiplexer has been switched. For

this reason, it is recommended to discard the first converted value after a channel change and to repeat the call (Atmel 2015).

- Assumes only the lowest system clock of 16MHz, this corresponds

$$16MHz \times 104 \approx 1664$$

Assuming only the lowest system clock of 16MHz, this corresponds to 1664 clock cycles. The repeated call, which is needed for the sequential query of all analog pins cost another 104µs (or another 1664 clock cycles). Since a "RISK" processor (as used for Arduino) usually require only a single clock cycle for the execution of a command it could execute 3200 commands, rather than waiting for that time elapse.

- A further disadvantage is that the sample moment between the channels is not only single but double conversion time due to the facts that the first sampled channel has to be discarded to get the input voltage of the AD stable.

3.5 APPLICATION COMMUNICATION AND MASS STORAGE OF DATA

For Supervisory Control and Data Acquisition (data logging), some Arduino can be supplemented by so-called shields to an Ethernet interface. Also there are ARDUINO versions with on-board integrated Ethernet hardware. For access from OSI-Application layer a library is available. This can be extended by use of a library with very basic functions for web-applications. A web server based on this basic functionality is part this project and used for all application communication and opened up some additional debug and service options.

Due to the 2k Byte extremely limited random access memory, it is neither suitable for a comfortable web interface nor for data logging. In comparison, the capacity of a modern SD card is nearly unlimited; however, the access speeds are orders of magnitude less. Therefore all mass data as html-encoded web pages are outsourced to the SD card. To keep the option of showing updated content of variables in the GUI a parser was implemented to replace specially code fragments with values of content from the RAM.

4 APPROACHES TO GET AN ACTUAL VALUE

The main objective of the project is to control the excitation current of the synchronous machine in order to vary its terminal voltage (Output voltage). More specifically the amplitude of all three phases or one of these corresponding alternatives to determined the replacement value for the measure of the three phases. Therefore, it has to be seen that

- the ADC of the Arduino can handle unipolar signals in the range of 0V to a maximum of 5V,
- while the output of the generator is three phase AC system with amplitudes of
  - $\hat{U}_{PP} = 230V \cdot \sqrt{2}$ between the outer conductors or
  - $\hat{U}_{PN} = 230V \cdot \frac{\sqrt{2}}{\sqrt{3}}$ from each outer conductor to neutral

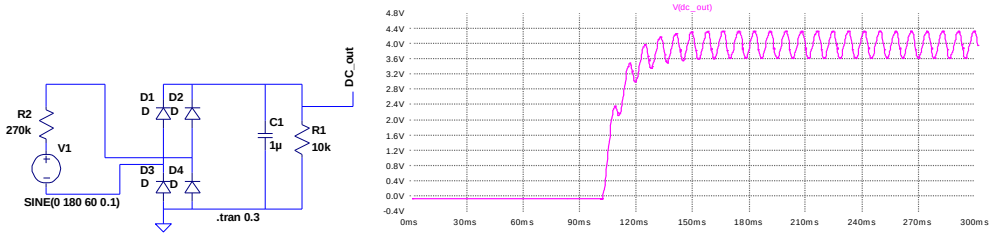
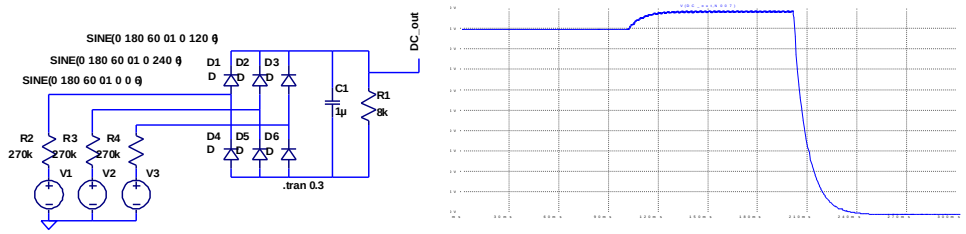
A variety of approaches has been envisaged. A small selection only briefly sketched with pros and cons, or just enumerated in the form of bullet points. Some approaches have been addressed and pursued separately from individual team members.

- One obvious way is to buy ready made RMS terminals. (Proven, readily available components = Low development risk)
  - Directly detectable output
    Protection for personnel and electronics by protective separation
  - But high conversion delay (about 100 to 200ms)
    Comparatively large design
    high financial cost, starting at about 250 € per channel
  - Suitability for fast control is limited with this option.
- With one, or a set of small transformers is also an option for voltage scaling and reliable separation, however, this can only be used in closets selection of transfer ratios, and particularly for small transformers a no linear transfer function may results which is detrimental for direct evaluation for the measure of the three phases.
- A capacitive divider would have advantages in terms of power dissipation, but was rejected due to the large component tolerances.

- A resistive divider directly connected to the Arduino inputs without galvanic isolation was also proposed, developed, tested and proved satisfactory. Although, potential separation must be issued with free option to measure either the interlinked voltage or the phase voltage.

For further treatment of signals different rectifier circuits were simulated using *spike software Table 4-1*.

Table 4-1: Simulation Result with Different Rectifier Circuits Using LTspike Software


B2 rectifier schematic: Clearly visible are two negative effects. A slowly change of mean and the remaining ripple. For the time constant, a compromise must be chosen, since the influence on the two effects is diametrical.

B6 rectifier schematic: In this circuit the ripple is significantly lower at similar time constant. Reducing the time constant to the principle of conditional inertia and the ripple occurs again clearly.

5 PULSE WIDTH MODULATION (PWM)

PWM is a method of obtaining analog results by digital means (Arduino 2016).

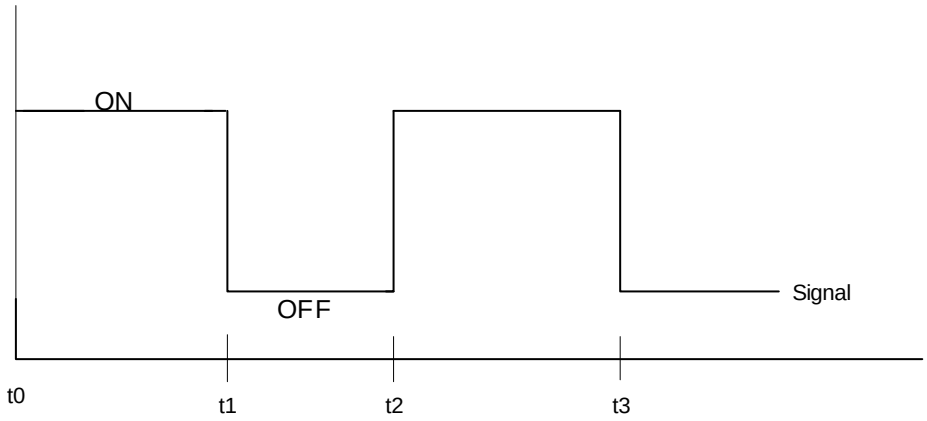


Figure 5-1: PWM Signal.

As shown in *Figure 5-1* the duration the signal is ON is known as pulse width. To obtain different analog values the pulse width is changed or modulated (Arduino 2016). Varying the pulse width, the average voltage is varied and this process is used to control power electronics.

5.1 DUTY CYCLE

As shown in *Figure 5-1*, the duty cycle is the percentage of ON time for one period (Wikipedia 2016). It is obtained from how long it takes for the pulse to be on and this is used to calculate the voltage. From the *Figure 5-1*,

$$1\ period = t_2 - t_0 = (ON + OFF) = T \quad (1)$$

$$Duty\ cycle = \frac{t_1 - t_0}{T} \times 100 \quad (2)$$

$$Duty\ cycle = \frac{ON}{(ON + OFF)} \times 100 \quad (3)$$

To get the required voltage:

$$Voltage = \frac{Duty\ cycle}{100} \times Peak\ Voltage\ from\ the\ PWM \quad (4)$$

In this project, Arduino is used and the *peak voltage from the PWM* output pins is 5V. Therefore, as the duty cycle is varied (percentage ON) the output voltage for switching the power electronic (IGBT) is also varid resulting in varying the current.

5.2 PWM FREQUENCY

PWM frequency determine the switching frequency for the power electronic components (IGBT) used for the excitation board. To control the switching frequency, the PWM frequency needs to be controlled using prescaler. This gives full control of the switching frequency for the power electronic components (e.g: IGBT) used. Increasing the switching frequency reduces the ripples and decrease the size of the filters used, however, this also has a drawback due to switching losses. Arduino PWM pin 3 is used in this project which is assigned by default to timer2, therefore, to vary the PWM frequency on the pin Timer2 control register B is used (TCCR2B). As shown in *Table 5-1* the PWM base frequency for *pin 3* and *pin 11* is 31250Hz, this is divided by different prescaler values shown in the table to obtain the disired PWM frequency.

Table 5-1: Default PWM Pins for Different Timers, the PWM Base Frequency for the Pins and the Prescaler Values (Wikispaces 2016), (Atmel 2015).

	Arduino PWM Pins	Arduino PWM Pins	PWM Base Frequency (Hz)	Prescalers values	
Timer0	5	6	62500	1	$PWM_{Freq} = \frac{PWM\ Base\ Freq(Hz)}{Prescaler}$
Timer1	9	10	31250	8	
Timer2	3	11	31250	64	
				256	
				1024	

6 TIMER INTERRUPT

An interrupt is a signal transmitted to the process by the hardware or software to indicates an event that requires immediate attention. It causes an interruption of the execution of the current code perform by the process to attends to the high priority condition (Wikipedia 2016). The timer interrrupts enables a task to be performed at every specific timed intervals irrespctive of what is going on in the code. The idea behind using a timer interrupt in this project is to enable sampling of the three phase voltage signals (R, S, T) from the Arduino analog inputs at a defined specific intervals.

Arduino has onboard crystal oscillator that oscillates at 16MHz on which all timing is based on as shown in *Figure 3-1*.

In this project, Arduino Uno, Arduino Ethernet and Leonardo are used and these are based on ATmega 168A/328P (Arduino 2016) with three interrupt timers.

- Timer0
- Timer1
- Timer2 (Atmel 2015)

Each of these timers can exist in three different modes in addition to timer1 which has additional interrupt capture event (Atmel 2015). In this project, only timer0 is used and the discussion will be based on that. To enable full control of the timing for the sampling of the data and other control strategies timer0 interrupt was developed with different control register settings without using any library that acts like a black box.

6.1 TIMER0

Timer0 is an 8-bit Timer/Counter Control Register (TCCR0) with Output Compare Registers (OCR0A and OCR0B) that enables accurate program execution timing (Atmel 2015). It has three interrupt vectors:

- Timer/Counter0 Compare Match A
- Timer/Counter0 Compare Match B
- Timer/Counter0 Overflow

6.2 TIMER0 CONTROL REGISTERS

The timer0 has two control registers, two output compare registers, Timer Counter and interrupt mask register. Each of these are 8-bit registers which determine the different modes of operation for the interrupt, how often the interrupt is called and enabled. The control registers are briefly described in *Table 6-1*.

Table 6-1: Control Registers for Timer0 (Atmel 2015).

Registers	Brief Description
TCCR0A (Timer/Counter0 Control Register A)	Determine different modes for the interrupt.
TCCR0B (Timer/Counter0 Control Register B)	Determine different modes for the interrupt
OCR0A (Output Compare Register0 A)	Use for compare match when Timer/Counter0 Compare Match A interrupt vector is used
OCR0B (Output Compare Register0 B)	Use for compare match when Timer/Counter0 Compare Match B interrupt vector is used
TCNT0 (Timer CouNT0)	Timer counter for Timer0
TISM0	It enables the interrupt for Timer0

Each contents of the registers in *Table 6-1* are respectively shown in *Table 6-2* to *Table 6-7*.

Table 6-2: TCCR0A - Timer/Counter0 Control Register A (Atmel 2015).

7	6	5	4	3	2	1	0
COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00

COM0A – Compare Output Mode for Timer0 channel A: Its settings determing whether the Output Compare pin (OC0A) channel A is set, clear or toggled (Atmel 2015).

COM0B – Compare Output Mode for Timer0 channel B: Its settings determing whether the Output Compare pin (OC0B) channel B is set, clear or toggled (Atmel 2015).

WGM0 – Wave Generation Mode for Timer0: Its setting determine the interrupt mode (Atmel 2015).

Table 6-3: TCCR0B - Timer/Counter0 Control Register B (Atmel 2015).

7	6	5	4	3	2	1	0
FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00

FOC0A/B – Force Output Compare for Timer0 channel A: Its settings is determined by COM0A/B (Atmel 2015).

CS – Clock Select: It determine the clock source to be used by the Timer/Counter by division by prescaler (Atmel 2015).

Table 6-4: OCR0A - Output Compare Register A (Atmel 2015).

7	6	5	4	3	2	1	0
OCR0A(7:0)							

Table 6-5: OCR0B - Output Compare Register B (Atmel 2015).

7	6	5	4	3	2	1	0
OCR0B(7:0)							

Table 6-6: TCNT0 - Timer/Counter Register (Atmel 2015).

7	6	5	4	3	2	1	0

Table 6-7: Timer/Counter Interrupt Mask Register (Atmel 2015).

7	6	5	4	3	2	1	0
-	-	-	-	-	OCIE0B	OCIE0A	TOIE0

- *OCIE0A – Timer/Counter0 Output Compare Match A Interrupt Enable:* Setting the bit to 1 enables the interrupt vector for Timer/Counter0 Compare Match A (Atmel 2015).
- *OCIE0B – Timer/Counter0 Output Compare Match B Interrupt Enable:* Setting the bit to 1 enables the interrupt vector for Timer/Counter0 Compare Match B (Atmel 2015).
- *TOIE0 – Timer/Counter0 Overflow Interrupt Enable:* Setting the bit to 1 enables the interrupt vector for Timer/Counter0 Overflow (Atmel 2015).

The timer0 can operates in different interrupt modes determined by the Wave Generation Mode (WGM) settings in the TCCR0A or TCCR0B Register as shown in *Table 6-8* (Atmel 2015).

Table 6-8: Timer0 Interrupt Modes (Atmel 2015)

Timer0 Interrupt Modes	WGM0 Settings	Brief dDscription
Normal Mode	0	Counter counts up to max (0xFF: 255) and resets to 0
CTC (Clear Timer on Compare Match)	2	The output of the counter (TCNT) is continously compared with the Output Compare Register (OCRA or OCRB). A match is used to generate an interrupt.
Fast PWM	3 (when TOP is defined as 0xFF) 7 (For OCR0A)	The PWM that goes as fast as the clock will allow. Not synchronized with the timing clock.
Phase Correct PWM	1 (when TOP is defined as 0xFF) 5 (For OCR0A)	PWM that is synchronized so that it is symmetric with respect to the timing clock

6.3 INTERRUPT FREQUENCY

In this project, timer0 interrupt is used to start the ADC conversion for the three analog channels connected to the three phases of the three phase generator at a fixed interval. How often the interrupt is called (*Int. frequency*) is determined by *equation 5*.

$$Int.frequency = \frac{Clock\ Frequency}{P\ \times\ (OCR0A + 1)} = \frac{16\ 000\ 000}{P\ \times\ (OCR0A + 1)} \qquad (5)$$

$$OCR0A = \frac{Clock\ Frequency}{P\ \times\ Int.frequency} - 1 = \frac{16\ 000\ 000}{P\ \times\ Int.frequency} \qquad (6)$$

where:

$Clock\ Frequency$ - is the frequency of the crystal oscillator (*16MHz*) for Arduino board.

P - is the prescaler: *1, 8, 64, 256 and 1024 (Atmel 2015).*

To control the sampling frequency, the timer0 interrupt frequency needs to be controlled. As shown in (5) the interrupt frequency depends on the value of the *prescaler (P)* and OCR0A. And the value of the *prescaler (P)* chosen is dependent on the clock select (CS0n) from the TCCR0B Register. The harware automatically determine which value of the prescaler based on the Clock Select (CS02:0) bit set to 1 in the TCCR0B. *Table 6-9* shows clock select bits and the corresponding division by prescaler that would be selected by the hardware.

Table 6-9: Clock Select Bit Description and their Corresponding Prescaler Values (Atmel 2015).

Bit	CS02	CS01	CS00	Prescaler Value
0	0	0	0	No clock source
1	0	0	1	1 (No prescaling)
2	0	1	0	8
3	0	1	1	64

4	1	0	0	256
5	1	0	1	1024
6	1	1	0	External clock source on T0 pin on falling edge
7	1	1	1	External clock source on T0 pin on rising edge

For 60Hz generator project, the timer0 interrupt operates on CTC mode which is briefly described below.

6.4 CTC INTERRUPT MODE

In CTC (Clear Timer on Compare Match) the Timer Counter (TCNT0) counts from zero up and compare with the values in the OCR0A, if the value in the OCR0A matches the Timer Counter value, an interrupt is triggered. The interrupt frequency used in this project is 1KHz. This is obtained from equation (1) by setting the OCR0A to 255 and prescaler (P) to 64 (i.e: setting *CS00 = 1 and CS01 = 1* in TCCR0B Register). The settings use for the CTC mode and frequency setting is shown in *Table 6-10* to *Table 6-12*.

Table 6-10: Timer0 Control Register Settings for Timer Interrupt Implemented in the Project (Atmel 2015).

CTC Mode	OCR0A	CS02	CS01	CS00	This results in prescaler of
WGM01 = 1	255	0	1	1	64

Table 6-11: TCCR0A Settings for the Timer0 used in the Project (Atmel 2015).

COM0A1	COM0A0	COM0B1	COM0B0	-	-	WGM01	WGM00
0	0	0	0	-	-	1	0

TCCR0A = 2.

Table 6-12: TCCR0B Settings for the Timer0 used in the Project (Atmel 2009).

FOC0A	FOC0B	-	-	WGM02	CS02	CS01	CS00
0	0	0	0	0	0	1	1

TCCR0B = 3.

TIMSK = 2.

6.5 ADC (ANALOG TO DIGITAL CONVERSION) INTERRUPT VECTOR

From the findings shown in *Figure 6-1* it takes about 104 to 116us to read an analog values from Arduino analog pin using the *anaolgRead()* function. Based on this findings, to read the three analog channels connected to the three phases generator will required 312 to 348us and this would limit usability for fast control. The code for the findings is shown in *Appendix 7*.

```
---Results---
Analog Read Value: 716  Analog Read Exec Time: 212 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
Analog Read Value: 716  Analog Read Exec Time: 116 us
```

Figure 6-1: Execution Time for AnalogRead() Function

Based on this findings, ADC interrupt was developed to overcome this challenge.

The ideas behind ADC is that the Arduino hardware has the ability to start an ADC conversion with less time and then trigger an interrupt when the conversion is complete. This implies that other tasks can be executed simultaneously while the conversion is ongoing and when the conversion is complete the analog value is fetched from the ADC Register (Sweeney 2012) before the next conversion is started.

To realize this, the various control registers for the ADC were set as shown in *Table 6-13* and *Table 6-14*.

Table 6-13: ADCSRA Control Register (Atmel 2015).

7	6	5	4	3	2	1	0
ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
1	1 (by Timer0 ISR)	0	0	1	1	1	1

- *ADEN – ADC Enable bit:* Sets to 1 to enable ADC and 0 to disable the ADC.
- *ADSC - ADC Start Converstion bit:* Set to 1 to start ADC conversation and reset to 0 when the conversation is ready.
- *ADATE – ADC Auto trigger Enable (Not used for the project):* Auto start the conversation.
- *ADIF - ADC Interrupt Flag:* Sets to 1 when the conversion is complete and 0 otherwise.
- *ADPS2:0 – ADC Prescaler bit:* Sets all the bits to 1 for 128 ADC prescaler value (default ADC prescaler is 128). Therefore, the ADCSRA register is set to

ADCSRA = 207.

Table 6-14: ADMUX Register (Atmel 2015).

7	6	5	4	3	2	1	0
REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0
0	1	0	0	0	0	0	0

- *REFS1/REFS0* – Voltage Reference Selection bit for ADC.
- *REFS0:* Set to 1 for AVCC (5V) reference voltage.
- *ADLAR – ADC Left Adjust Result:* Set to 1 to left adjust and 0 to right adjust. Arduino has 10-bit resolution and will required 16-bit register to store the analog values. The ADLAR bit determine if 8-bit or 16-bit register is needed. Therefore, set ADLAR to 0 (16-bit register) (Atmel 2015).

- *MUX4:0 – ADC Multiplexer channels:* Set to 1 to change the analog channel. This is set manually in the ADC ISR vector. Therefore the ADMUX Register is set to: *ADMUX = 64* (With subsequent changing of the MUX channel in the ADC ISR).

Once the conversation is completed, the ADC Interrrupt Flag (ADIF) is set to 1 to trigger the ADC interrupt and the converted values fetched from the ADC ISR (Atmel 2015).

7 EXCITATION POWER ELECTRONIC

The excitation power electronic used to used vary the excitation current of the machine consists of:

- IGBT (Insulated Gate Bipolar Transistor)
- Gate driver
- Diode
- Resistor
- Bipolar capacitor.

A IGBT is an insulated gate transistor designed for fast switching (Wikipedia 2016). They are fast switches with high voltage and current capabilities commonly used when fast switching is required. When voltage is applied to the gate G, It allows current to flow when switch on (On state) and stops the current flows when switch off (Futureelectronics 2016). It works by applying voltage to the gate and combines the characteristic of MOSFET (Metal Oxide Semiconductor Field Effect) with high current and low saturation capability of BJT (Bipolar Junction Transistor) (Wikipedia 2016). The switching process of IGBT is similar to that of MOSFET and the working principle is shown in *Figure 7-1*. It works like a variable resistor controlled by the gate voltage for switching high current. It is associated with Collector (Vcc), the Gate (G) and the Emitter (E).

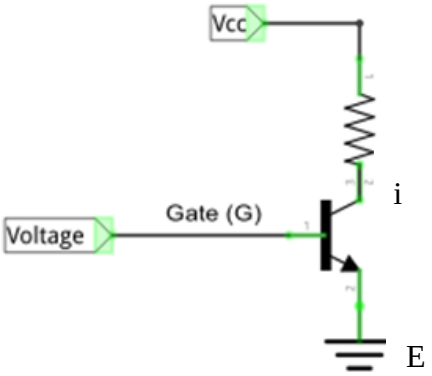


Figure 7-1: An Overview of an IGBT Working Principle.

As the voltage is applied between the Gate (G) and the Emitter, current is allowed to flow between the collector (Vcc) and the emitter (E).

Depending on the voltage applied between the Gate and the Emitter, the resistance between the Vcc and the E varies. when the applied voltage at the gate decreases, the resistance between the Vcc and the E increases blocking the current from flowing between the Vcc and the E. As the applied voltage at the gate is increased, the resistance between the Vcc and E decreases allowing current to flow.

For the 50/60Hz synchronous machine, the excitation current is varied to change the output voltage. The excitation winding consists of a coil as shown in

Figure *7-2*. The inductance *L* shown in the figure represent the excitation winding. The *G, R, S, T* represent the machine and its phase to phase output voltages at the clamp. The Zener diode *D* and resistor *Rr* are used to protect the IGBT from over voltage and current during switching off.

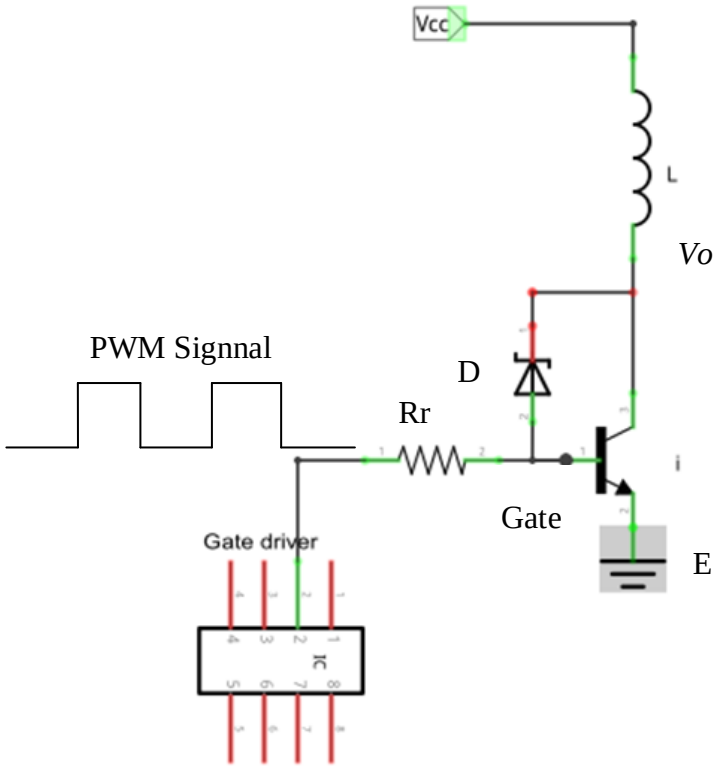


Figure 7-2: Connection Overview for the 60Hz Machine with the Excitation Board.

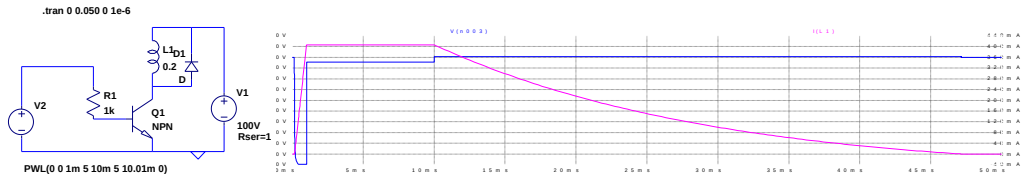
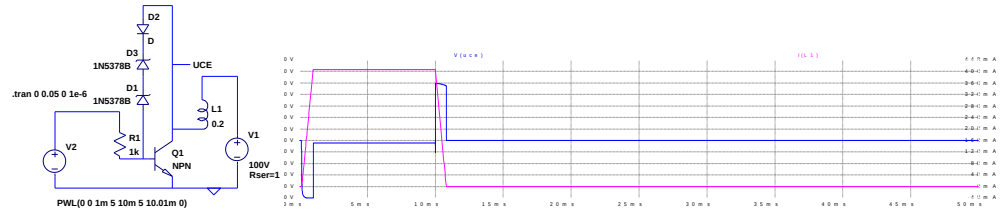
As the input voltage (PWM Signal) to the gate is varied, the current through the coil (L) to the E is varied also. At the moment the voltage is applied to the gate, the impedance between the Vcc and E decreases in proportion to the gate voltage, the current through through the coil to E increases with increase in the gate voltage causing the voltage between the Vcc and and E to decrease. When the gate voltage is off, the IGBT opens, the high current in the coil leads to high voltage *(Vo*) at the input of the IGBT. To protect the IGBT from the high voltage, the Zener diode *D* opens in reverse direction and the current through it leads to a voltage drop across the resistor *Rr* which charges the gate capacitor at the gate input of the IGBT. This Zener diode ensures a constant voltage *Vo* at the input of the transistor and a linear drop in current when the gate voltage is off. As the current through the excitation coil *L* changes the output voltages (*R, S, T*) of the 50/60Hz machine *G* changes and this process is used for the control of the output voltage of the machine.

7.1 SIMULATION OF PWM FOR INDUCTIVE LOAD

The circuit diagram for the Excitation board in *Appendix 2* was designed by *Helge Lorenzen* for the project. The simulation of the circuit is shown in *Table 7-1*.

When switching inductive loads, circuitry arrangements for the protection of the transistor must be taken. Without any protection circuit the voltage over inductance increases rapidly over all limits, which destroys the transistor immediately. Therefore it is common recommended to add Freewheeling diode, often in series with a damping resistor for less slowly drecrease of the current through the coil. The voltage over the damping resistor leads to an increase of voltage stress, depending on its value and the current in the switch off moment (Helge 2016). The function of decrease stays exponential. However, the circuit uses can withstand voltage of the transistor rarely, and not during the whole discharging process. More effective is the discharging with a circuit as shown in *Table 7-1*.

Table 7-1: Simulation Results for Slow / Rapid Discharging of Current for the 60Hz Excitation Board. The Circuit Diagram for the Simulation was Designed By Helge For the Project.


Freewheeling diode for inductive load, here without damping resistor. *Advantage:* Minimal voltage stress for the transistor *Disadvantage:* The slowly exponential decay of load current

Collector voltage cutoff circuit, *Advantages:* Rapid excitation reduction. Adjustable voltage stress for the transistor (note the collector voltage doubled but safely limited) *Disadvantages:* At the same PWM frequency greater riple so that the PWM frequency should be increased as long the increase of losses is exceptable.

Both options were considered as assembly variants for the project. *Figure 7-3* shows the picture for the excitation board. The board was designed by *Helge Lorenzen* for the project.

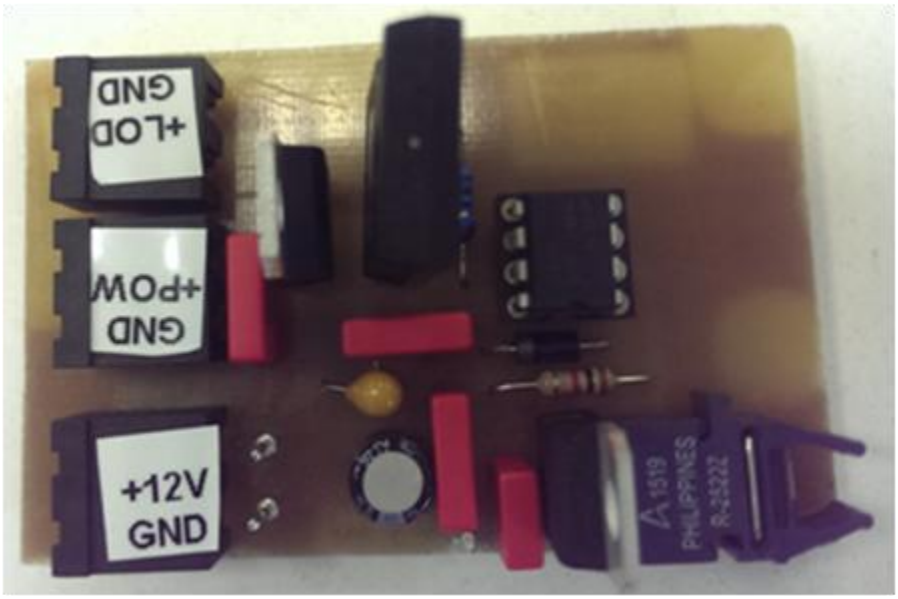


Figure 7-3: Excitation Board for the 60Hz Machine Designed By Helge and Built By Me

The values for the various components used for the excitation board are shown in *Appendix 2*.

8 RESISTIVE VOLTAGE DIVIDER MEASUREMENT APPROACH

The measurement electronic consists of three voltage dividers each connected to the three phases of the 60Hz synchronous machine for downscaling the phase voltage. For simplicity, only one of the voltage dividers for the three phases is considered in *Figure 8-1*. The various parameters in the figure are defined in *Table 8-1*.

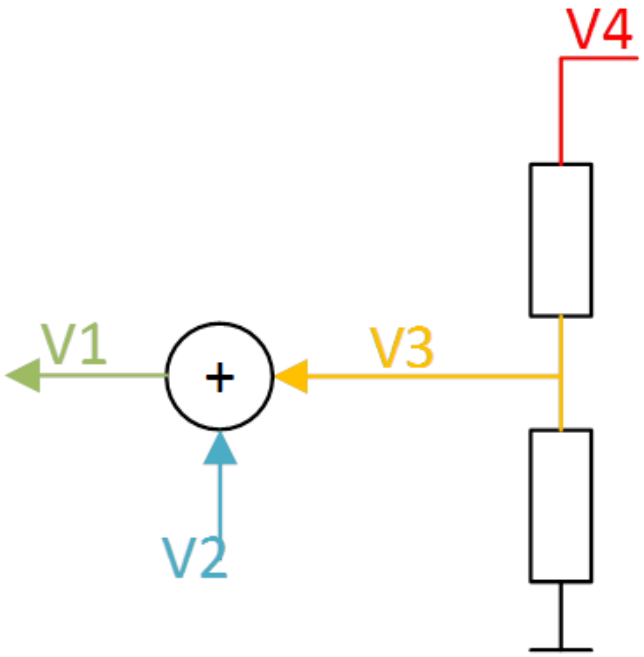


Figure 8-1: One of the Single Phases of the Measurement Electronic for Downscaling the Nominal Phase Voltage of the three Phase 50/60Hz Generator

Table 8-1: Definintion of Various Variables Used for the Voltage Divider of the Measurement Electronic

V1	V2	V3	V4
Input to Arduino Analog Pin from the voltage divider	DC Offset added to the *V3*	Output of the voltage divider in AC	Input to the Measurement Electronic from 3P Generator

Each of the offset output V1 from the divider are feed into Arduino analog pins (*A0, A1 and A2*) to be sampled by Arduino ADC. *Figure 8-2*shows the connection to the

Arduino analog pins. In the figure each of the three voltage dividers (representing *R*, *S*, *T*) from the measurement electronic are connected to the analog inputs of the Arduino.

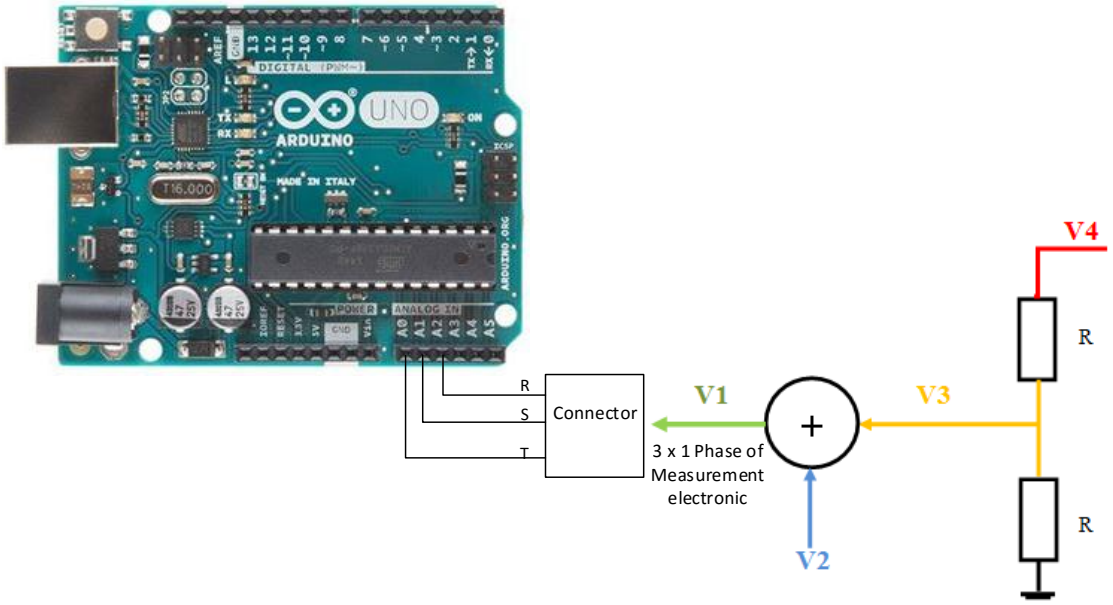


Figure 8-2: Measurement Approach for the Phase Voltage of the 50/60Hz Generator

8.1 MEASUREMENT ELECTRONIC

The measurement electronic and the PCB were designed by *Helge Lorenzen* for this project. The circuit diagram and the PCD are shown in *Appendix 3*. *Figure 8-3* shows the hardware photo mounted on the power cabinet.

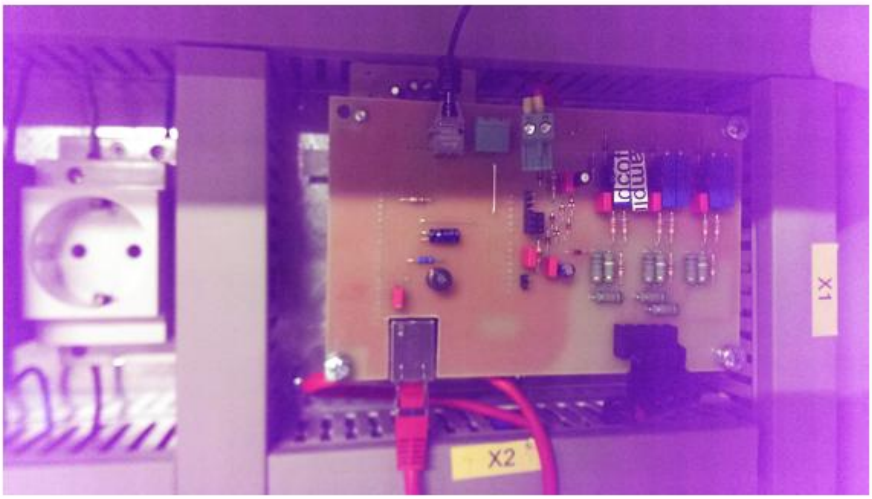
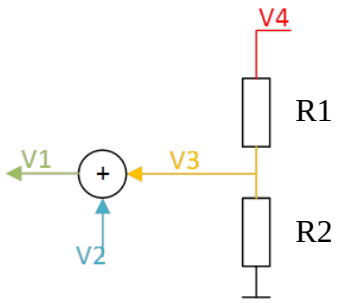


Figure 8-3: Measurement Electronic Board Mounted in the Power Cabinet with Arduino Board Connected At the Back.

8.2 SIGNAL CONDITIONING (VOLTAGE DIVIDER HARDWARE)

From the voltage divider shown V3 is obtained as shown in *equation 7*.



$$V_3 = \frac{R_2}{R_1 + R_2} V_4 = \frac{1}{K} V_4 \qquad (7)$$

Where

$V_3 = 2.5Vac,$

$V_4 = 250V\ (phase - to - phase\ voltage\ for\ the\ 50/60Hz\ Generator)$

$$\frac{1}{K} = \frac{R_1 + R_2}{R_2} \qquad (8)$$

Therefore, the ratio

$$\frac{R_2}{R_1 + R_2} = \frac{V_3}{V_4} = \frac{1}{100} \qquad (9)$$

This implies that R_2 will be 1% of the total sum of the resistors ($R_1 + R_2$) and R_1 will be 99% of the total sum of the resistors in the divider circuit.

The R_2 and the DCOffset (not shown in the figure) consist of variable resistors (potentiometers) for adjusting the amplitude of V_3(ac) and DCOffset values to 2.5Vac and +2.5Vdc respectively and this is shown in the circuit diagram in *Appendix 3*.

The resistor values and other components used are stored in the SD Card attached to the cover of this report.

For the signal conditioning of the masurement electronic the values in *Table 8-2* were adopted for the voltage divider of the measurement electronic. The DCOffset is included in the hardware design for the measurement electronic.

Table 8-2: Signal Conditioning of the Measurement Electronic.

	Value (Arduino Analog value)	Value – 512	V1 (V2 + V3)	V2 (Constant)	V3 (V4/K)	V4
Max. Value	1023	511	5V	2.5V	+2.5V	+268V
Value at 0	512	0	2.5V (*peak-to-peak*)	2.5V	0V	0V
Min. Value	0	-512	0V	2.5V	-2.5V	-268

Table 8-2 shows the expected Analog values from Arduino based on the input voltage V1 from the measurement electronic. The V3 (2.5Vac) is offset by the DCOffset value of +2.5V and this is implemented in the measurement hardware because Arduino analog pins can only measure a positive voltage from 0-5V. As a result, to measure the

negative value of AC voltage, the following scalings in *Table 8-3* were implemented in the hardware.

Table 8-3: AC output voltage of the Voltage Divider and the Corresponding Arduino Analog Value

Arduino Analog Value	V3 (Vac)
1023	+2.5Vac
512	0
0	-2.5Vac

8.3 HARDWARE CALIBRATION

The Matlab program of *Appendix 4* was developed and used for the calibration of the hardware. The DC voltage was used for the calibration with the following steps:

- Connect the three outputs (V1) of the measurement electronic to Arduino Analog pins.
- Connect the three inputs of the measurement electronic to DC voltage and set the voltage to 0V.
- Observe the analog values from Arduino and tune the DCOffset potentiometer until the analog value is 512.
- Set the DC voltage to +268V, measure the corresponding analog values for R, S, T and input the values into Matlab code in *Appendix 6* in the variable ‚*RSTRawAnalogVal* ‘ for the positive measure of voltage.
- Set the DC voltage to -268V and input the corresponding analog values into the Matlab code in the variable ‚*RSTRawAnalogVal* ‘ for the negative measure of voltage.
- Set the variable ‚*DCInVoltage*‘ to 268 volts in the Matlab code and run the Matlab script. Observe the ‚*VoltperDigit*‘ (the Gain). The *VoltperDigit* for the three phases should be the same and should be close to 1: 0.9371; 0.9371; 0.9371. If the *VoltperDigit* for any of the phases is less, adjust the

corresponding potentiometer for the amplitude (V3) for the phase and observe the gain. Repeat the process until all the gains (*VoltperDigit*) are equal and close to one.

8.4 MEASUREMENT AND SCALING FOR VOLTAGE SETPOINT

The data in *Table 8-4* is obtained from the measurement electronic to find the behaviour of the measurement device to the input voltage in order to determine the range for the linearity of teh measurement device.

Table 8-4: Measurement Data From the Measurement Electronic.

DC Voltage	-300	-250	-200	-150	-100	-50	0	50	100	150	200	250	300
R	16	27	74	182	292	401	512	625	734	842	951	992	1004
S	16	27	74	182	292	401	512	625	734	842	951	992	1004
T	16	27	74	182	292	401	512	625	734	842	951	992	1004

The data from *Table 8-4* were ploted using Matlab in *Appendix 5* to obtain the plot in *Figure 8-4*. From the plot, it was seen that the measurement device shows linearity behaviour in the range: -200V to 200Vdc after which nonlinear behaviour sets in. The nonlinear behaviour observed resulted from the diodes and capacitors implemented at the of the voltage dividers to protect Arduino analog input from over voltage (spikes).

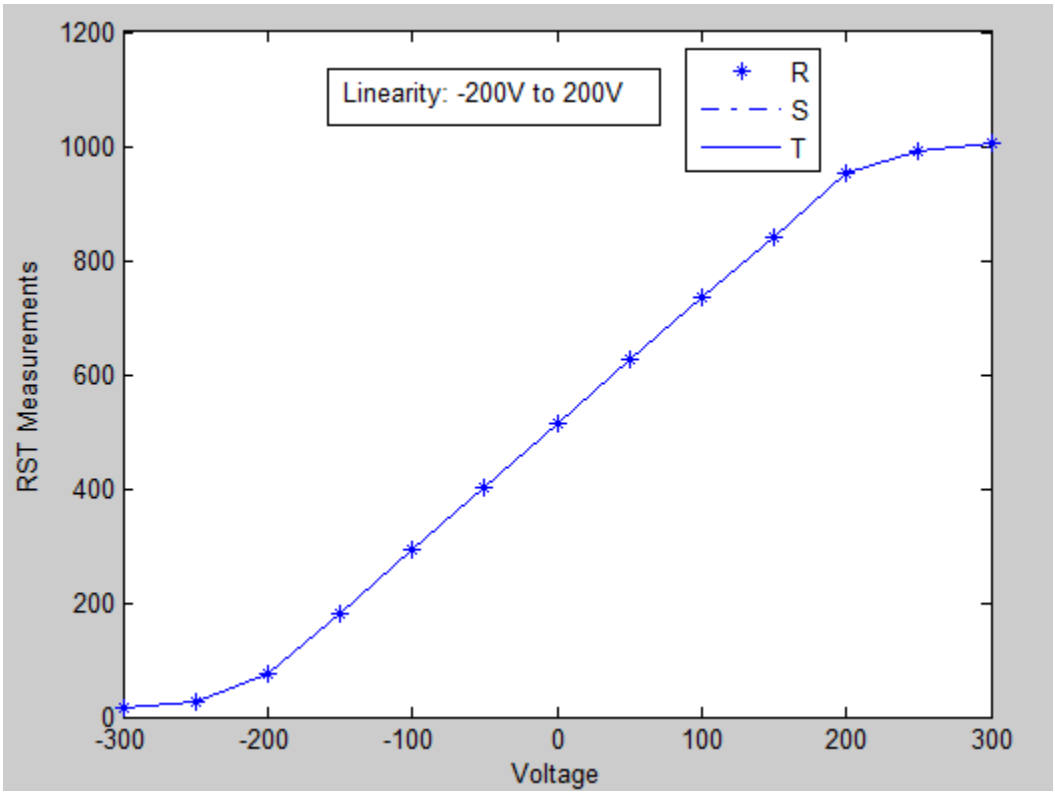


Figure 8-4: Measurement Electronic Showing Non-Linearity after 200VDC

For the sinusoidal, symmetrical and time-invariant system, the 3 phase is shown in *Figure 8-5.*

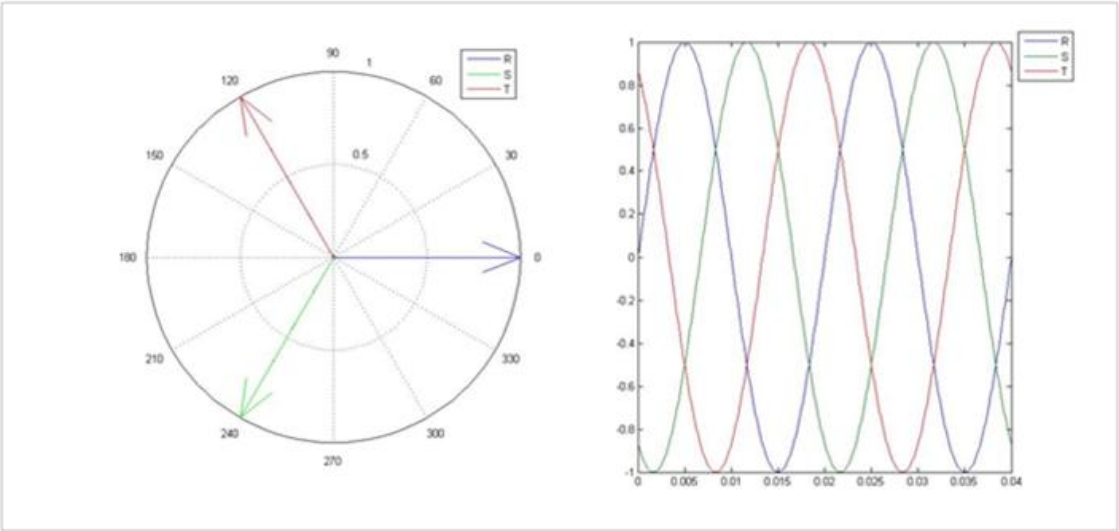


Figure 8-5: The three Phase Voltage for the Generator.

Table 8-5: Measurement Data Use for Scaling.

	Value (Arduino)	V2 (Constant)	V3 (V4/K)	V4
Max Value	951	+2.5V	2.5V	200V
Value at 0	512	+2.5V	0V	0V
Min. Value	74	+2.5V	-2.5V	-200V

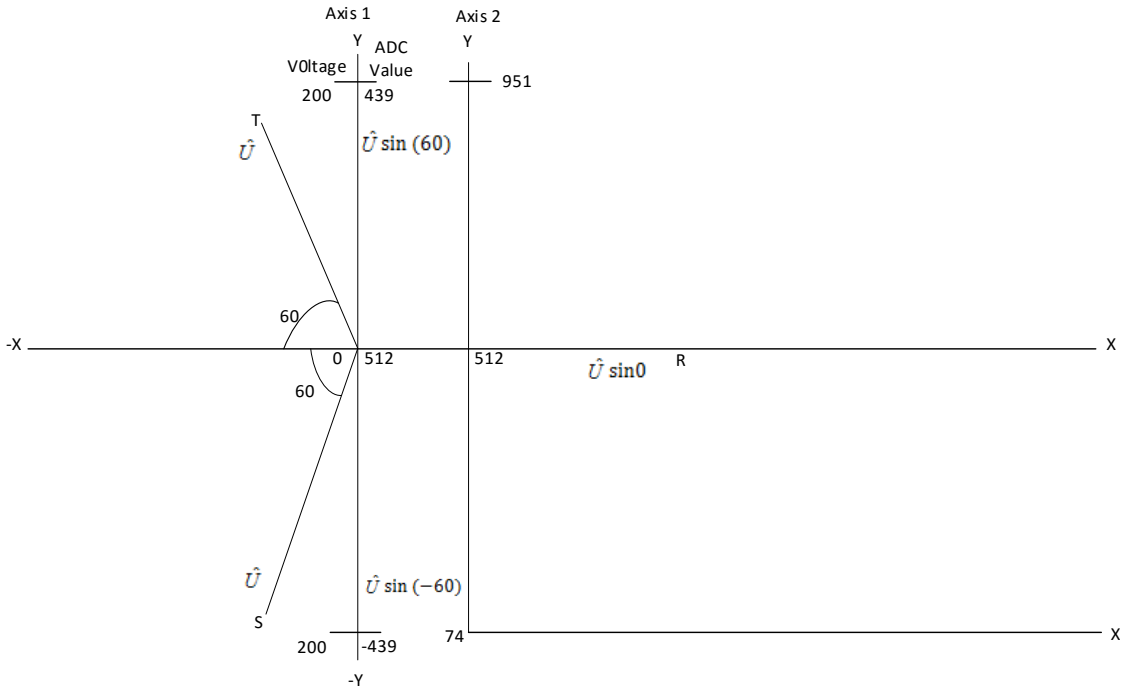


Figure 8-6: Scaling to Find the Measure for the Voltage Setpoint for the 50/60Hz Synchronous Generator.

In *Table 8-5* and *Figure 8-6* it could be seen that:

$$K = \frac{\max(V_4)}{\max(V_3)} = \frac{\min(V_4)}{\min(V_3)} = \frac{200V}{2.5V} \quad (10)$$

$$ADC\ Value = \left(\frac{u_X(t=0)}{K}\right).\left(\frac{439}{2.5V}\right) + 512 \quad (11)$$

$$ADC\ Value = \left(\frac{u_X(t=0)}{200}.\ 439\right) + 512 \quad (12)$$

Where $where\ u_X(t=0)$ is the instantaneous voltage for each of the 3 phases R, S, T.

$u_X(t=0)$ is evaluted and obtained in equation 14, *15* and *16*.

Figure 8-6 shows the phasor representation of *R, S, T*. The amplitude for each phases is denoted as $\hat{U}$ for a sinusoidal, symmetrical and time-invariant system.

From the figure $u_X(t=0)$ for each of the phases is calculated and used in equation 3.

Resolve each phases R, S, T to the horizontal and vertical axis of *Figure 8-6* in order to find the intantaneous voltage $u_X(t=0)$ for each phases:

Amplitude $\widehat{U}$ is given as:

$$\widehat{U} = \frac{U_D.\sqrt{2}}{\sqrt{3}} \tag{13}$$

Where U_D is the phase – to – phase voltage output voltage of the 50/60Hz synchronous generator.

For phase R:

$$\mathrm{u_R(t=0)} = \widehat{U}\sin 0 = \frac{U_D.\sqrt{2}}{\sqrt{3}}.0$$

$$\mathrm{u_R(t=0)} = 0U_D \tag{14}$$

For phase S:

$$\mathrm{u_S(t=0)} = \widehat{U}\sin(-120) = -\widehat{U}\frac{\sqrt{3}}{2}$$

$$= -\frac{U_D.\sqrt{2}}{\sqrt{3}}.\frac{\sqrt{3}}{2} = -0.7071.U_D$$

$$\mathrm{u_S(t=0)} = -0.707U_D \tag{15}$$

For phase T:

$$\mathrm{u_T(t=0)} = \widehat{U}\sin(-240) = \widehat{U}\frac{\sqrt{3}}{2}$$

$$= \frac{U_D.\sqrt{2}}{\sqrt{3}}.\frac{\sqrt{3}}{2}$$

$$\mathrm{u_T(t=0)} = 0.7071U_D \tag{16}$$

Where

$$sin0 = 0$$

$$\sin(-120) = -\frac{\sqrt{3}}{2}$$

$$\sin(-240) = \frac{\sqrt{3}}{2}$$

$$U_D \; is \; the \; phase - to - phase \; voltage \; for \; the \; generator$$

Substituting equation 14, *15*, and *16* into *12* for each phases:

For phase R:

$$adcValueR \; = 512 \qquad (17)$$

For phase S:

$$adcValueS \; = \left(-\frac{0.7071 U_D}{200}.\; 439\right) + 512 \qquad (18)$$

For phase T:

$$adcValueT \; = \left(\frac{0.7071 U_D}{200}.\; 439\right) + 512 \qquad (19)$$

U_D is the phase to phase voltage setpoint for the three phase generator specified by the user from the Graphical User Interface (GUI). This U_D specified by the user is converted in the program to the corresponding analog values for the three phases using *equation 17,18, 19* and then to analog setpoint in *equation 20* for the controller.

From equation *17*, *18*, and *19*, the setpoint for the controller will be:

$$Setpoint = [(adcValueR\,)^2 + \,(\,adcValueS)^2 + \,(\,adcValueT)^2] - 3 \times 512^2 \quad (20)$$

where

$$adcValue = ADC\;Value = \left(\frac{\mathrm{u_X(t=0)}}{200}.\; 439\right) + 512$$

In *equation 20*, the DC-Offset for the three phases implemented in the measurement hardware is removed from the setpoint. This is because the DC-Offset is also removed in the computation for the measured value of RST in the program.

To display the output measurement from Arduino to GUI in voltage (phase-to-phase) reverse calculation is required as shown in equation *21* - *23*.

$$u_{(t=0)} = \left(\frac{200}{439}\right)(ADC\;Value - 512) \qquad (21)$$

$$U_D = \frac{\sqrt{3}}{\sqrt{2}}.u_{(t=0)} \qquad (22)$$

$$U_D = 1.2247 u_{(t=0)} \quad (23)$$

Where U_D is the phase-to-phase voltage for each phases and $ADC\ Value$ is the analog value for each phases ffrom the Arduino analog pins.

8.5 EFFECTS OF REMOVING DC-OFFSET IN THE COMPUTATION

The idea behind removing the DCOffset in the computation is that:

- When the voltage from the 3 phase system is *0V* the corresponding measure for the 3 phases will also be 0. The computation for the measured value of the three phases implemented in the sketch (program) is shown in *equation 24*.

$$measured = [(R)^2 + (S)^2 + (T)^2] - 3 \times 512^2 \quad (24)$$

- When the voltage from the synchronous generator is maximum 200V, the corresponding measured will be *1,926,771*.
- When the voltage is -200V, the analog value will be 0 and the corresponding measure will be: -(3×512^2) as shown in *Table 8-6*.

Table 8-6: Effects of Removing the DC-Offset by Software.

	200V	0V	-200V
Analog Values	Assume: R = S =T =951 (Max voltage)	512	0
Measured value	$[(R)^2 + (S)^2 + (T)^2] - 3 \times 512^2$	0	-(3×512^2)

For a sinusoidal symmetrical time invariant three phase system this is not a problem as the three phases cannot be negative at the same time at any time instance due to the rotation of the angle and if they ar negative is just for that moment. But for a non three phase system for example DC voltage, if the three inputs to Arduino analog pins becomes negative, this will result to a negative measure of -(3×512^2). If the setpoint voltage U_D (*in this case: DC peak voltage*) is positive, the controller function will

always experience an offset (error) of $-(3 \times 512^2)$ when compare the setpoint with the ‚*measured value*' and it will try to eliminates this error but will not be able, resulting into saturation of the control signal.

- Therefore to use this project for all kinds of voltages (DC, phase voltage: phase-Neural, or phase-to-phase voltage: Line-to-Line) , the *DC-Offset* should be set to zero from the *HMI* in the *System status (DCOffset = 0).*

8.6 ACCESSING THE 3P GENERATOR THROUGH RS232

To overcome the challenge of debugging and uploading of sketch (Arduino program) without having to go to the 60Hz machine installed at the basement, *RS232 to Serial communication hardware for the 6 pin serial programming header* communication for Arduino Ethernet was designed by *Helge Lorenzen*. The circuit for the hardware is shown in *Appendix 6.* To realize this, an Ethernet cable (CAT 5) was used. The cable design is shown in *Figure 8-7*, the hardware and its connections with the cable and Arduino is shown in *Figure 8-8*.

The cable is only valid for Arduino Ethernet and any other Arduino products with 6 pin serial programming header for communication. The cable configuration and design in *Figure 8-7* is also stored in the SD Card in a folder with filename: *Excitation Electronic Documents.*

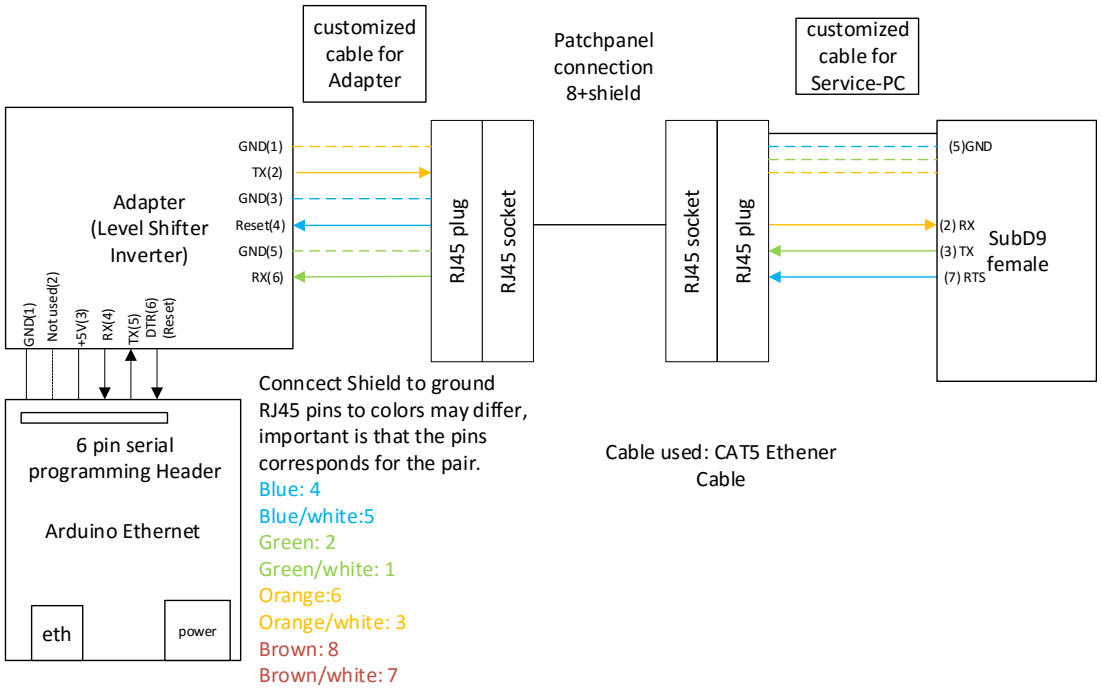


Figure 8-7: Ethernet Cable (CAT 5) Design For the Communication Between RS232 and the Arduino Ethernet 6 Pin Serial Programming Header.

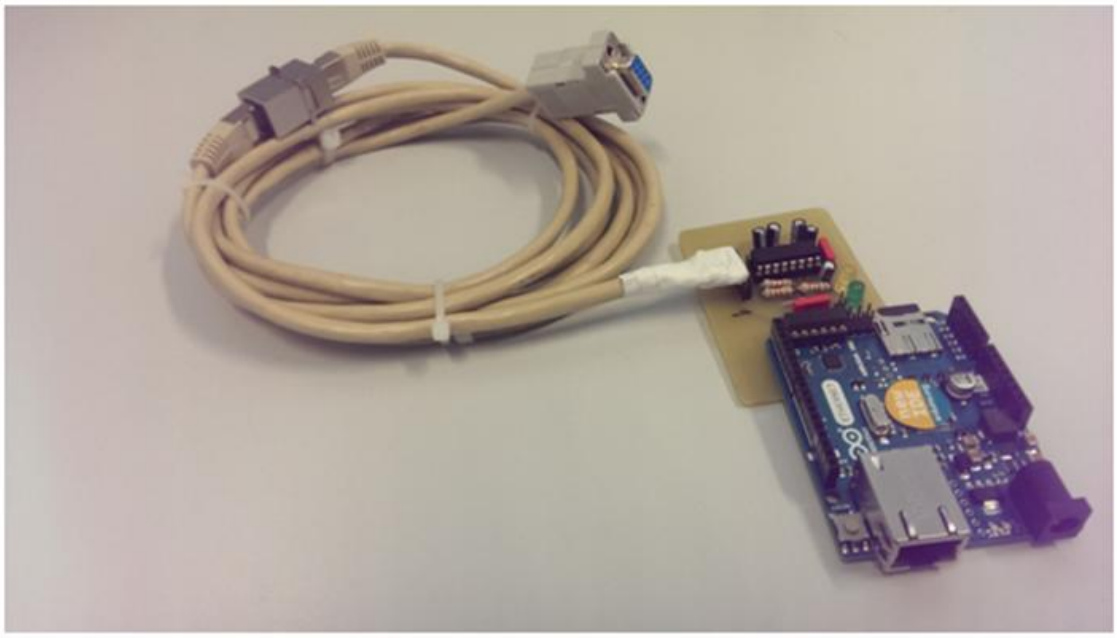


Figure 8-8: RS232 to Serial Communication Hardware Connected with its Cable and Arduino Ethernet.

To communicate with the 60Hz generator via the cable, connect the labtop to the RS232 port or via RS232 to USB cable and launch a terminal program ‚*Hterm*‘ to enable the connection to the serial port through the cable. This can be download from the link: http://www.circuitlake.com/serial-terminal-program-hterm.html

9 IMPLEMENTATION FOR PRELIMINARY INVESTIGATIONS

Preliminary investigations and comparison of measurement data by team members were carried out using different measurement approaches. The team members are:

- Helge Lorenzen - Project Manager
- Achema Egbunu - Master Student
- Tim Hanekamp - Wobben Research and Development staff
- Christian Fellensiek - Wobben Research and Development staff

9.1 DATA AQUISITION

Different measurement approaches were assigned to the team members under the supervision of *Helge Lorenzen*.

- *Helge Lorenzen* - PC Based measurment system for the Lab.
- *Achema Egbunu* - Resistive Voltage Divider with DCOffset and ADC Interrupt Vector
- *Tim Hanekamp* - Channelwise board with external ADC harware and op-circuits for signal conditioning.
- *Christian Fellensiek* - Transformers and op-circuits for signal Conditioning.

9.2 PC BASED MEASUREMENT SYSTEM

The PC based measurement system for the laboratory is a high quality data aquisition system for sampling data in the lab. This was used in order to compare its measurement with others. The measurement data obtained from this were plotted using MATLAB shown in *Figure 9-1*. In the figure, the first plot represnts the measurements from the PC based measurement system carried out by the Project Manager, *Mr. Helge Lorenzen*.

9.3 TRANSFORMERS AND OPERATIONAL AMPLIFIER-CIRCUITS FOR SIGNAL CONDITIONING

In this approach the phase voltage is step down to 24Vac and some operational amplifiers to further scale the voltage to the Arduino input recommended voltage (*0-5V*). The disadvantage with this approach is that loss of non-linearity may be expected. The measurement obtained with this approach was not pleasing, hence, result could not be use for comparison.

9.4 CHANNELWISE BOARD WITH EXTERNAL ADC HARWARE AND OP-CIRCUITS FOR SIGNAL CONDITIONING

This approach consists of three resistive voltage divider each contained in different boards (i.e. one resitive divider for one PCB) with external ADC. Each resistive voltage divider is connected to the each of the three phases of the 60Hz machine to scale the voltage to recommended input voltage for the external ADCs. The three external ADCs each connected to the output of the divider simultaneously sample the three phase of the machine, buffered the values into one of the ADC and finally transfer the data from the buffered ADC to Arduino via clocking. The advantage with this approach is that the three phases of the machine are sampled at the same time without any delay between the channels, hence less ripple is expected. The disadvantage is that it is more expensive and occupies more space due to three boards used for each phases of the machine.

The measurement data obtained from this were plotted using MATLAB shown in *Figure 9-1*. In the figure, the second plot represnts the measurements from this approach carried out by the *Tim Hanekamp*.

9.5 RESISTIVE VOLTAGE DIVIDER WITH OFFSET GENERATION AND AC ANALYSIS IN TIME DOMAIN WITH ARDUINO ADC INTERRUPT VECTOR

This approach consists of three resistive voltage dividers connected to each of the phases of the machine to scale the voltage to the recommended input voltage of the Arduino analog pins (*0-5V*). The three voltage dividers are implemented on one board with DC-Offset. Because Arduino has only one ADC to sample data from multiple analog channels, a delay between each channel is expected. To minimze this, ADC interrupt vector was developed to sample the data from the analog pins (*A0 –A2*) connected to each output of the three resistive voltage divders. The measurements were carried out from zero duty cycle (Machine PWM) to the maximum duty cycle of the machine (Max. Capacity of the Machine) in order to observe the behaviour of measurement systems.

The measurement data obtained from this were plotted using MATLAB shown in *Figure 9-1*. In the figure, the third plot represnts the measurements from this approach carried out by me.

9.6 COMPARISON OF FIRST FINDINGS

The data obtained from each approaches were plotted with MATLAB and comapred as shown in *Figure 9-1*. The MATLAB code for the plots is shown in *Appendix 8*. In order to minimize the number of pages, only two comparison are shown here. Detailed comparison with different PWM (0 - 255) is stored in the SD Card titled ‚*Measurement Report_html*‘, click on ‚RunMe.html‘ to view report. In the figure, three plots are obtained for each value of duty cycle.

- *First plot* - Helge Lorenzen (PC Based measurement system)
- *Second plot* - Tim Hanekamp (Resistive divider with External ADC)
- *Third plot* - Achema Egbunu (Resistive div. with Ard. ADC Int. Vector)

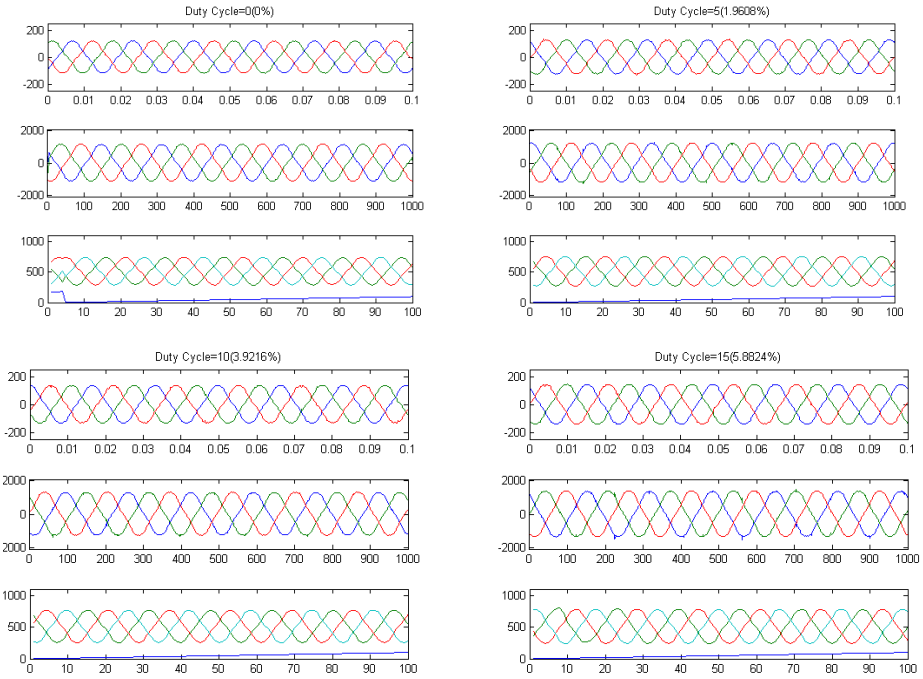


Figure 9-1: Comparison of the Measurements Obtained from Different Approaches.

From the plots stored in the SD Card, the behaviour observed from Tim Hanekamp plot (2nd plot) as the duty cycle increases was due to a signum errors in his measurement code which was later fixed. From *Figure 9-1* and the plots stored in the SD Card, it was seen that the measurement from Resitive voltage divider with DC-Offset and ADC Interrupt Vector (Carried out by Me) was very okay, however, some strange behaviour was seen at regular interval from the plots. This happens whenever there is a spike (High voltage greater than 5V) from the machine which is cutoff by the Arduino input. To address this problem and protect the analog input of the Arduino, a protective circuit consisting of Zener diodes and capacitor was implemented at the output of the voltage divider as shown in the circuit diagram in *Appendix 3.* A closer look at the measurements carried out by me shows some ripples which might be due to

- Delays from the ADC in sampling data from one channel to another
- The selection of reference voltage to Arduino input (AVREF)

In order to improve this challenge, external reference voltage supply was implemented on the measurement device shown in *Appendix 3* to provide reference voltage for the Arduino analog inputs.

10 DESCRIPTION OF THE SOLUTION FINALLY USED

- The measurement electronic circuit used is shown in *Appendix 3*.
- The excitation electronic circuit used is shown in *Appendix 2.*
- The RS232 to Serial communication circuit is is shown in *Appendix 6.*

The documents are also stored in the SD Card attached to the cover of this report in their respective folders.

10.1 SOFTWARE IMPLEMENTATION

The software for this project was designed as procedural and not as object oriented, hence, Object-Oriented, Analysis and Design was not used for the software design. Flow diagram and Star UML were used for the description of the program flow and the general overview.

10.1.1 ARCHITECTURE AND GLOBALLY OVERVIEW

The architecuture and the global overview for the software is shown in *Figure 10-1*.

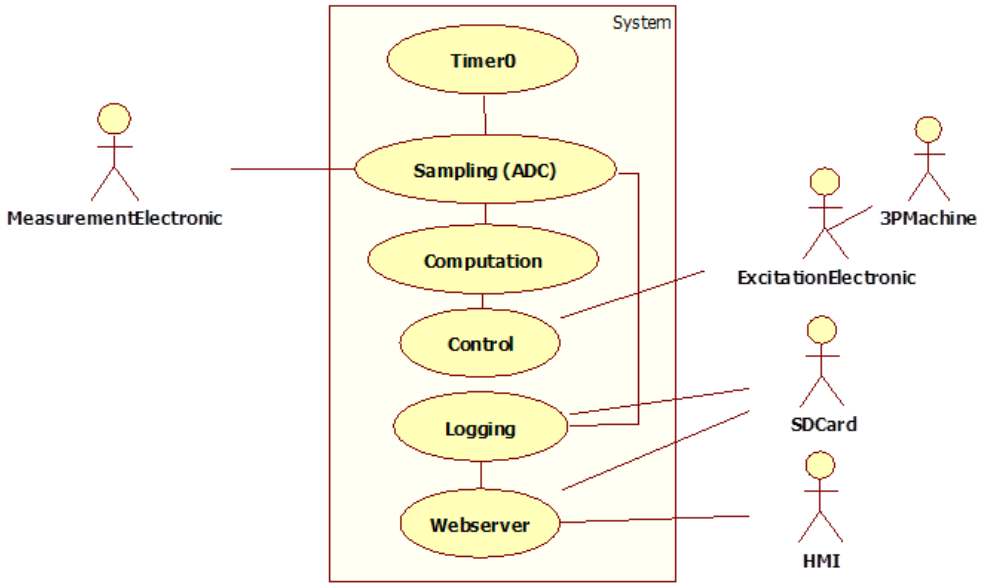


Figure 10-1: Program Overview.

10.2 DATA MANAGEMENT

In this project, all the functions declared access the data or store the data in the main program. The main program consists of three global arrays which all the functions can access. The global arrays are:

- *Global Byte Array – gArByte[] :* This stores only the data that are bytes (*i.e.:1 byte*).
- *Global Integer Array – gArINT[]:* This stores only the data that are integers.
- *Global Long Integer* – gArLong[]: This stores only the data that are long integers.

The structure of the main program with the global arrays and their respective variables definitions are shown in *Figure 10-2*. The various registers definitions and the program flow for the *void setup()* and *void loop()* are also shown in the figure. The structure of the main program shown in *Figure 10-2* are also stored in the SD Card in a folder with filename: *Program Structure.*

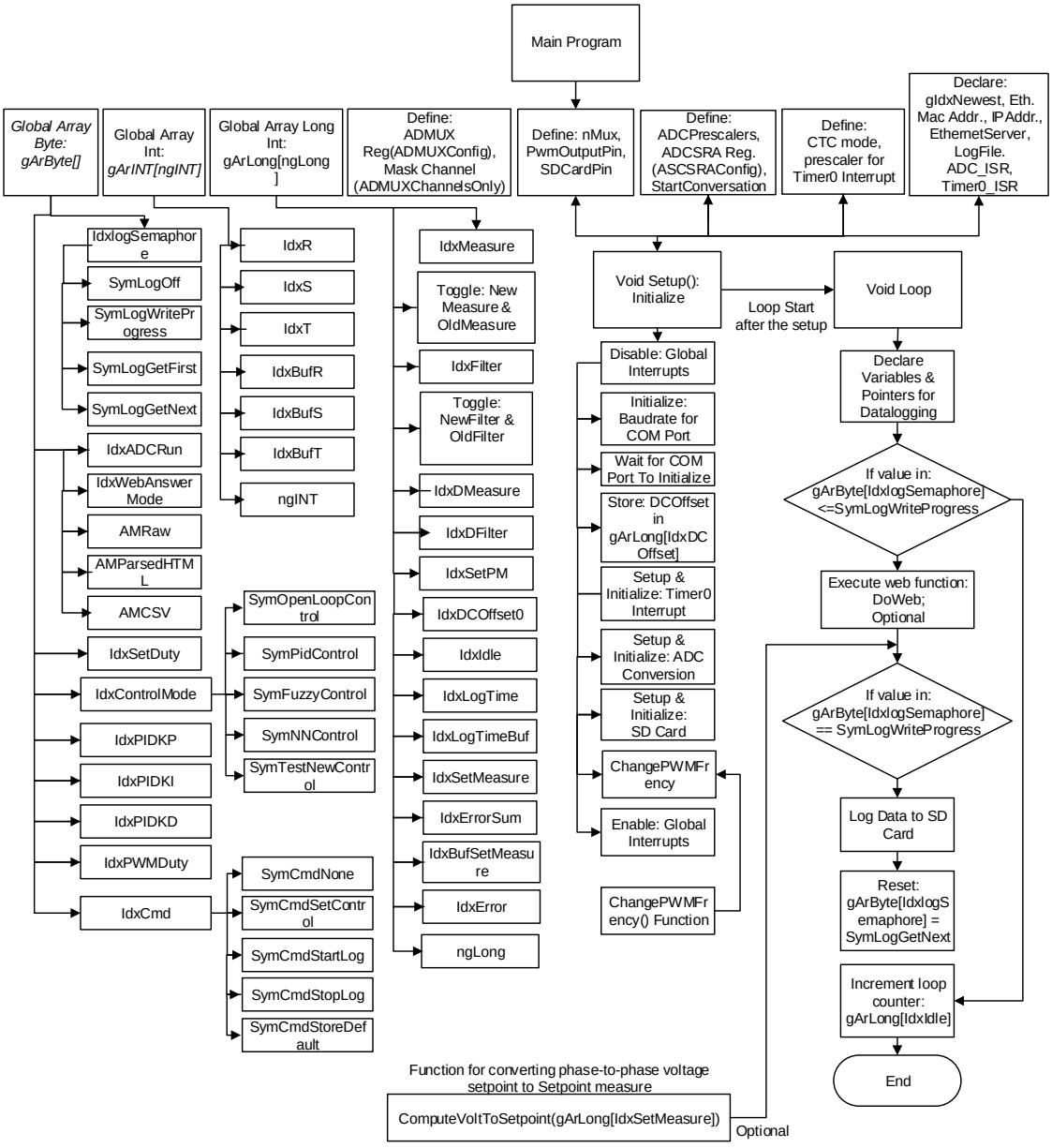


Figure 10-2: Program Structure for the Main.

As shown in the figure, the variables preceeded with *‚Idx‘* (index) represents the location in the array for the variables and those preceeded with *‚Sym‘* (Symbolic name) represents the the meaning of content of the array for the specific variable with ‚*Idx*‘. As shown in the global array byte (gArByte[]) in FIG H, the ‚*logSemaphore 0‘* is an *IdxlogSemaphore* define as 0 in the main program. The 0 represents the space in the array (Array index) that holds *IdxlogSemaphore* and this space can have a value of

- *0: SymLogOff*
- *1: SymLogWriteProgress*

- *2: SymLogGetFirst*
- *3: SymLogGetNext*

As shown in *Figure 10-3*. Other global arrays with their meaning and content value are shown in *Figure 10-2*.

Global Byte Array

Array index	Meaning of Index	place holder	Meaning of content of volatile byte gArByte[11]:
0	logSemaphore 0	0	LogOff 0; WriteProgress 1; GetFirst 2; GetNext 3
1	ADCRun 1	1	Channel just scanned
2	**WebAnswerMode**	1	**AMRaw 0; AMParsedHTML 1; AMCSV 2**
3	IdxSetDuty	0	Setpoint for open loop control
4	IdxControlMode	0	OpenLoopControl 0; PidControl 1; FuzzyControl 2; NN 3; Test 4
5	PIDKP	0	parameter for PID-Control
6	PIDKI	0	parameter for PID-Control
7	PIDKD	0	parameter for PID-Control
8	PWMDuty	0	backuped actual duty cycle (at PwmOutPin 3)
9	**IdxCmd**	0	**CmdNone 0; CmdSetControl 1; CmdStartLog 2; CmdStopLog 3 (set to 0 after execution)**
10	ControlTrigger actual value	192	count down for change of Duty cycle in control
11	ControlTrigger interval	0	sequence of Duty cycle in control (start value of count down)

Figure 10-3: Global Byte Array Index Definition and and their Respective Content Meaning

In the main loop the data logging is executed based on the value in the array space for *IdxlogSemaphore*. The void setup() setup and initialize the timer0 interrupt, ADC conversion, and SD card. The program code for the Main is shown in *Appendix 8*.

10.3 ADC CONVERSATION AND MULTIPLEXER

The program flow and timing sequence to the ADC_ISR() is shown *Figure 10-4*.

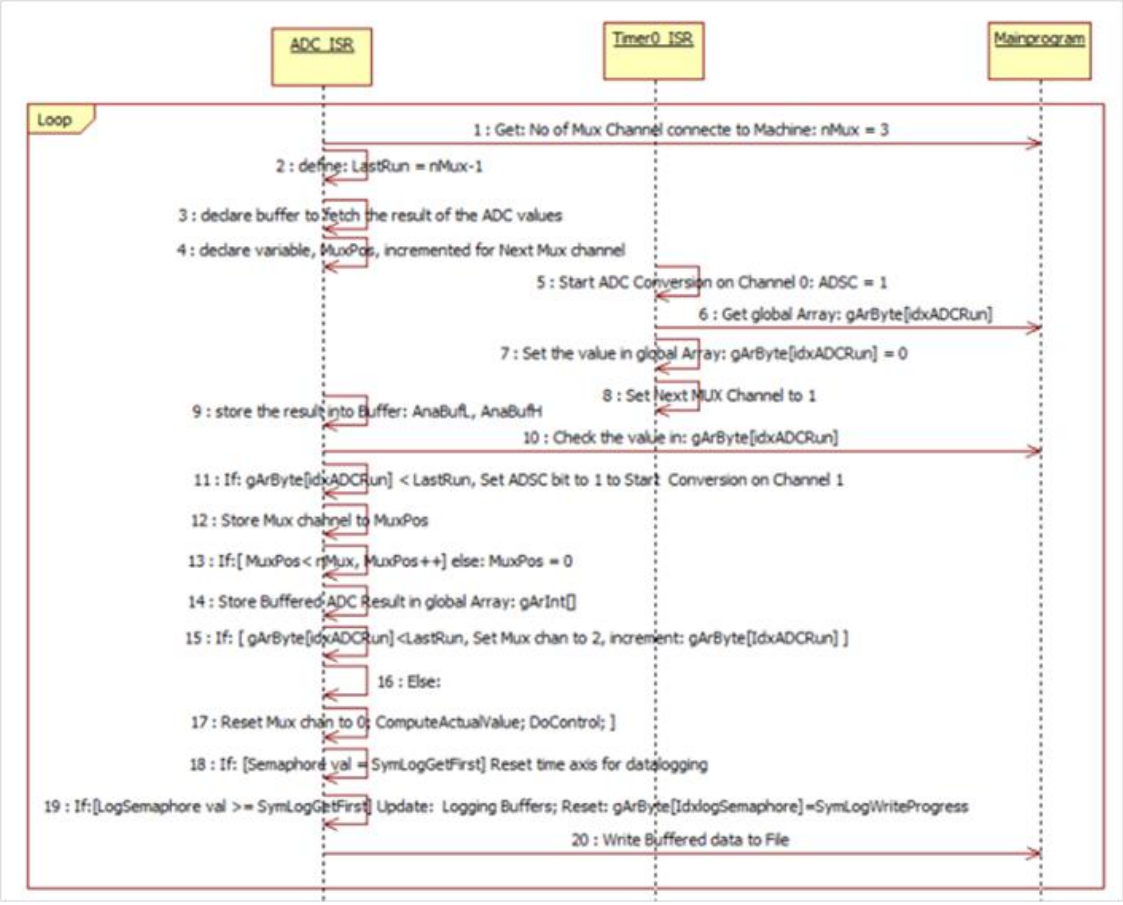


Figure 10-4: Program Sequence for the ADC_ISR Function.

As shown in *Figure 10-4* the number of Mux channel (nMux) connected to the 60Hz machine is 3 and this is defined as a global variable in the main program. The *LastRun* defined in the ADC_ISR function represents the actual Mux channel that is switched by the multiplexer since the Mux channel is zero indexed (counting from 0 to 2 = 3 channels). The value in the global Array *gArByte[IdxADCRun]* is used for timing control for starting the ADC conversion (i.e. setting ADSC to 1). The ADC convertion is implemented in manual mode with the Timer0 ISR starting the manual conversion (Setting ADSC to 1) at a specified frequency. The setup for the ADC is implemented in the *void setup()*. The sequence of operation is discussed in the following sections.

- The ADC prescalers are defined in the maian program, however, the default ADC precaler is 128 is used in this project and is defined as ‚*ADCPrescaler128 = 7*‘ in the main program.
- The ADCSRA Register is defined in the main program as:

ADCSRAConfig = 0xC8 + ADCPrescaler128 = 200 + 7 = 207 (In Dec.):
This sets *ADEN, ADSC, ADIE ADPS2, ADPS1,* and *ADPS0* bits in the ADSCRA Control Register to 1. As seen, the ADC Start Conversion bit (ADSC) is set to 1 from ‚*ADCSRAConfig'* defined.

- The ADMUX Control Register is defined in the main program as:
  ADMUXConfig = 0x40 = 64.
  This sets the ADC reference voltage (REFS0) bit to 1 in the ADMUX Register. This value is ‚*ANDed*' with the wanted MUX channel number ‚*ADMUXChannelsOnly'* to enhance the selection of the intended channels and mask out the rest bits.
- The mask bit ‚*ADMUXChannelsOnly'* is defined in the main program as:
  ADMUXChannelsOnly = 0x1F = 31 (In dec.).
  These registers are initialized in the *void setup()* in the main program based on the settings above.

```
ADMUX = ADMUXConfig;    // see prescaler options defined above
ADCSRA =  ADCSRAConfig;
ADCSRB =  0;
```

- After the conversion is started by the Timer0 ISR function, the value in the global Array ‚gArByte[IdxADCRun]' is reset to 0 for the timing control of ADC_ISR and also to stabilize the ADC voltage before the next conversion (Atmel 2009) and the next Mux channel is set to 1 as shown.

```
ISR(TIMER0_COMPA_vect)
  {
    ADCSRA |= ADStartConveration; // Start Conversion (for channel 0)
    gArByte[IdxADCRun]=0; // increase is done by ISR-ADC itself.
   /* Channel 0 was preselected in the last run off ISR-ADC
   *  so we started conversion of channel 0 here.
   *  With short delay (the one row above) after the start of the conversion
   * the MUX is switched to channel 1. This is important to get
   * the voltage stable when ISR-ADC starts the next conversion */
  ADMUX=ADMUXConfig+1;
  }
```

- Once the conversion is complete, the ADC_ISR() function is called based on the ADC Interrupt Flag and ADC Interrupt Enable bits (ADIF and ADIE) to fetch

the results of the conversion from the ADC register and then start the next conversion on channel 1 by resetting *ADSC* bit to 1.

- The result of the conversion is stored in a global array ‚gArIn[]' which is accessed by ‚*Computation()* ' function.
- Once the conversion on channel 1 is in progress the *ADC_ISR* increment the value of ‚*MuxPos* ' based on the condition shown as shown.

```
// ########### Determine the MUX channel to be set next  ################
MUXPos = ADMUX & ADMUXChannelsOnly; // Mask out the Rest
if (MUXPos < nMux) {
  MUXPos++;
} else {
  MUXPos = 0;
}
```

The *MuxPos* is incremented for two purposes. First is to stabilize the ADC voltage before the channel 2 is selected and also as Mux channel which is ‚*OR*ed' with the content of the *ADMUX* based on the condition shown.

```
if (gArByte[IdxADCRun] < LastRun)
{
  ADMUX= ADMUXConfig|MUXPos;   // 
  gArByte[IdxADCRun]++;        // 
  }
```

- Once the value in the global array ‚gArByte[IdxADCRun]' is equal to or greater than *2* (*LastRun*) then all the three phases (*R, S, T*) connected to the 60Hz generator have been sampled and stored in the global array and the following functions would be called.
  - *ComputeActualValue()*
  - *DoControl()*
- Once the Computation and the control function are executed, the value in the global array for log semaphore ‚*gArByte[IdxlogSemaphore]*' is checked to determine if the user has requested for datalogging or not. If the user has requested for data logging, the value in the array would be set to ‚*SymLogGetFirst*' *(i.e. 2)* which will cause the *ADC_ISR* to reset the log time and update the buffer with new data for logging.

The code for the ADC_ISR for sampling the RST phases is shown in Appendix 9.

10.4 COMPUTE ACTUAL VALUE

The program sequence for computing the actual value of the RST phases implemented in this project is shown in *Figure 10-5.*

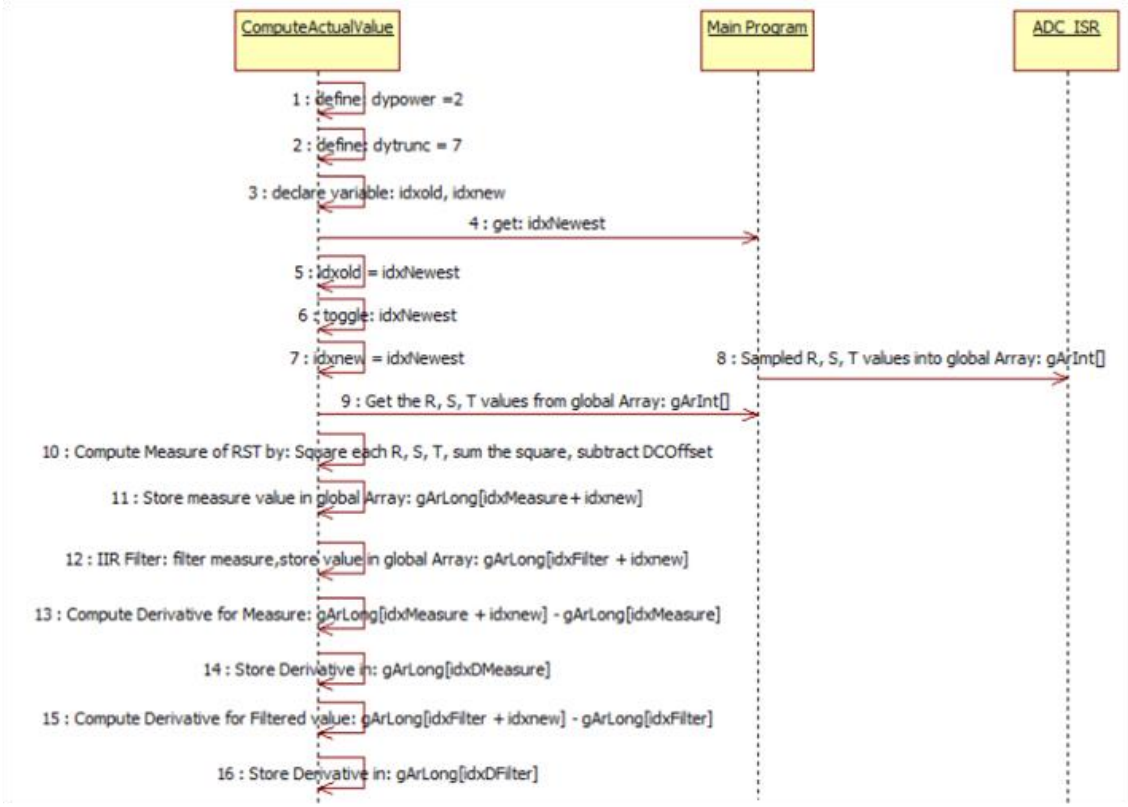


Figure 10-5: program Sequence for the Computing Actual Value of RST function (i.e.: ComputeActualValue()).

The program code implemented is shown in *Appendix 10.*

10.5 CONTROL ALGORITHMS

The equations for the *P, PI and PID control* implemented is shown in this section followed by the algorithm.

For P Controller:

$$u = K_p e \qquad (25)$$

$$e = r - y \qquad (26)$$

where:

u is the controller output.

K_p is the proportional gain.

e is the error.

r is the *setpoint and*

y is the output of the 60Hz machine.

PI Controller:

$$u = K_p e + \frac{K_p}{T_i}\int_0^t e(t)dt \qquad (27)$$

$$u = K_p e + K_i \int_0^t e(t)dt \qquad (28)$$

Where:

u is the controller output

K_p is the proportional gain

e is the errror, defined in *equation 26.*

$K_i = \frac{K_p}{T_i}$ is the gain for the integral part

$\int_0^t e(t)dt$ is the sum of the errors over time.

PID Controller:

$$u = K_p e + K_i \int_0^t e(t)dt + K_p T_d \dot{e} \qquad (29)$$

Using *euler forward* to discretize $\dot{e}$:

$$\dot{e} = \frac{e_{k+1} - e_k}{\Delta t} = \frac{r_{k+1} - y_{k+1} - (r_k - y_k)}{\Delta t} \quad (30)$$

Where the current setpoint r_{k+1} and the previous setpoint r_k are the same. That is

$$r_{k+1} = r_k = r \qquad (31)$$

Therefore, *equation 30* becomes:

$$\dot{e} = \frac{r_{k+1} - y_{k+1} - (r_k - y_k)}{\Delta t} = -\frac{y_{k+1} - y_k}{\Delta t} \qquad (32)$$

$$\dot{e} = -(y_{k+1} - y_k).f \qquad (33)$$

$$f = \frac{1}{\Delta t} \qquad (34)$$

where:

y_{k+1} is the current output measurement from the 60Hz machine.

y_{k+1} is the previous output measurement from the 60Hz machine.

Δt is the sampling interval which is provided by timer0 interrupt.

f is the sampling frequency which is the frequency of the timer0 interrupt.

Substitutes *equation 33* into *equation 29*:

$$u = K_p e + K_i \int_0^t e(t)dt - K_p T_d (y_{k+1} - y_k).f \quad (35)$$

$$u = K_p e + K_i \int_0^t e(t)dt - K_d (y_{k+1} - y_k).f \quad (36)$$

Where K_d is the gain for the derivative.

Therefore, *equation 36* can be written

$$u = P + I - D \quad (37)$$

where:

$$P = K_p e \qquad (38)$$

$$I = K_i \int_0^t e(t)dt = K_i \sum_0^t e(t) \qquad (39)$$

$$D = K_d (y_{k+1} - y_k).f \qquad (40)$$

10.5.1.1 CONTROL ALGORITHM

- Set the setpoint measure
- Compute the error: $e = r - y$
- Sum the errors
- Compute the error slope $\dot{e} = (y_{k+1} - y_k)$
- Determine the interrupt frequency f or fixed sampling interval Δt
- Multiply the error with the proportional gain: $P = K_p e$
- Multiply the error sum with the gain of the integral: $I = K_i \sum_0^t e(t)$
- Multiply the error slope with the derivative gain and the sampling frequency:

$$D = K_d (y_{k+1} - y_k).f$$

- Sum: $P + I - D$
- Store the value of the sum in a variable: $u = P + I - D$
- Adjust K_p set $K_i = K_d = 0$ for P Controller until the output is stable
- Adjust K_p and K_i and set $K_d = 0$ for PI Controller until the output is stable
- For the PID Controller, adjust K_i, K_i and K_d until the output is stable

Remark:

In order to have good resolution to tune the control parameter K_p, K_i and K_d and to be able to work with integer values when tuning the parameter the parameter are divided by a factor by shifting to the right (>>) as shown in *equation 41*. However, the parameters value are shifted after multiplying with their respective error, integral and derivative values to avoid shifing all the parameters values to zero.

$$u = \left((K_p e) \gg a\right) + \left(\left(K_i \int_0^t e(t)dt\right) \gg \mathrm{b}\right) - \left((K_d (y_{k+1} - y_k).f) \gg c\right) \quad (41)$$

where:

a, b, and c are the factors to be divided with.

Remarks:

Bit shifting (>>) operator for negative values in Arduino can leads to undefined behavior, therefore, the negative sign are taken into consideration in the code.

10.5.2 FUNCTION THAT WRITES CONTROL SIGNAL TO PWM OUTPUT PIN

The program flow for the function that write the output of the controller to the PWM output pin is shown in *Figure 10-6*.

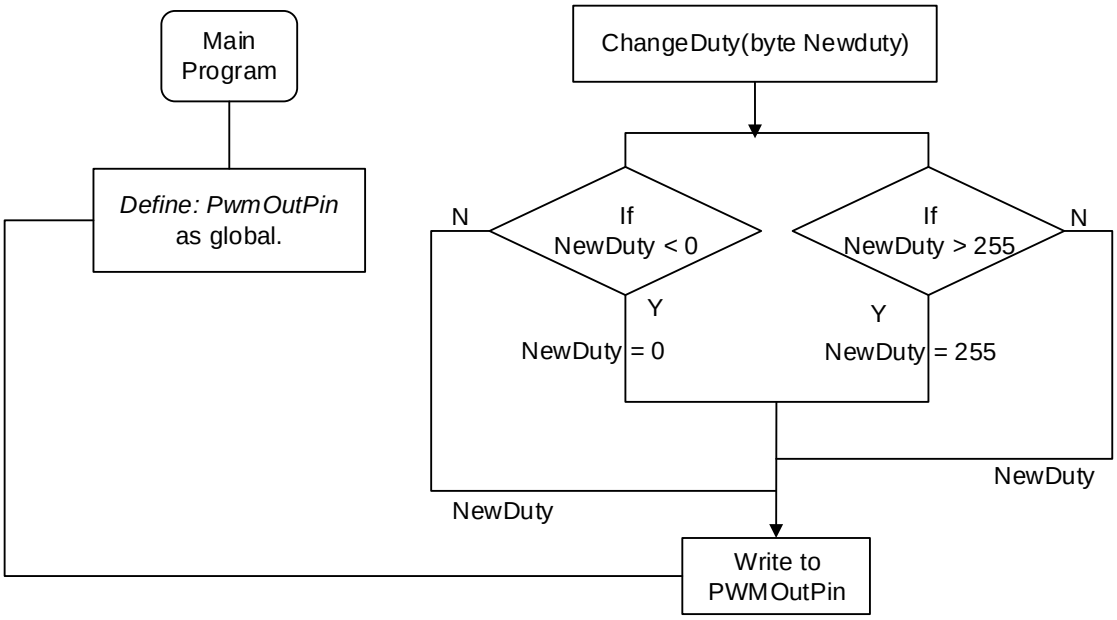


Figure 10-6: Prgram Flow for the Function that Writes Control Signal to PWM Output Pin.

This function is called by any of the control strategies to write the control output to the output pin (*PWMOutPin*). Within this function a bond is set on the control output (duty cycle) between 0 and 255 to prevent a negative values from being written on the output pin and also the controller signal from exceeding the maximum PWM value.

The program code for this function is shown in *Appendix 11*.

10.5.3 OPEN LOOP CONTROL

For the open loop control, the setpoint is set by the user as duty cycle (*range*: *0 - 255*) and the function ‚*ChangeDuty()*‘ is called in order to write the value to the *PWMOutPin 3*. The value supplied as duty cycle is stored in the global variable ‚*gArByte[IdxSetDuty]*‘ the function *ChangeDuty()* is called with an input argument

newDuty once the user click on open loop from the *HMI* and the value in the global array *gArByte[IdxSetDuty]* is written to the output pin.

10.5.4 P-CONTROL

The P Control in this project is implemented as a function which can be inserted into any other project based on the guidelines given in this section.

10.5.4.1 THE P CONTROL STRUCTURE

Figure 10-7 shows the the overview of the P controller implemented in this project.

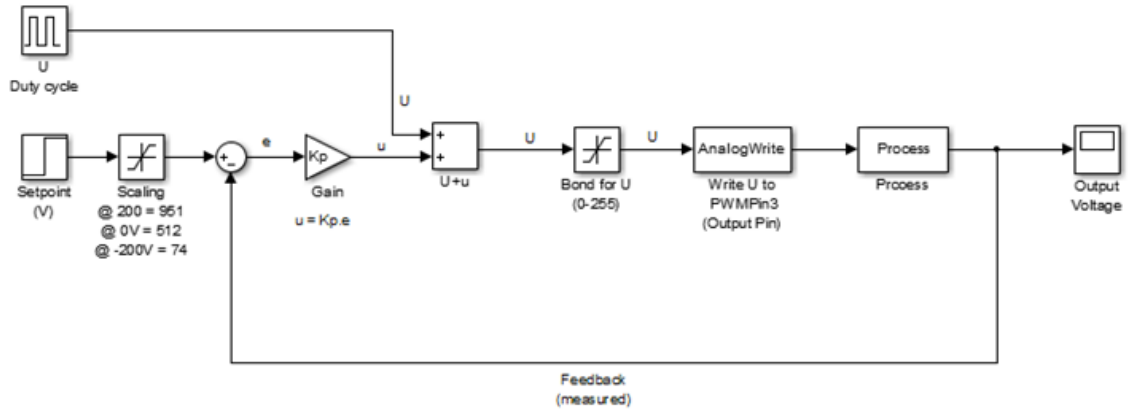


Figure 10-7: The Structure of the P Controller Function.

From the figure, *U* is the actual duty cycle and *u* is the control signal from the P Controller. The control signal *‚u‘* is added to the actual duty cycle *U* which increases or decreases the duty cycle based on the offset (error) between the setpoint and the measurement values (Output). The *U* is then written to the PWM output pin 3 on the Arduino where the process is connected to vary the excitation current for the process winding. The writing to the PWM output pin is done by calling a function *‚ChangeDuty()‘* responsible for that. It should be noted that the process in this case is referred to the 6Hz machine connected to the Arduino PWM pin 3 through the excitation electronic board.

Input to the controller function:

- *gArLong[idxSetPM];*
- *gArLong[idxFilter];*
- *PWM (Actual duty cycle) defined as: gArByte[IdxPWMDuty]*

- *Afunction call ‚ChangeDuty()‘ with an input argument ‚U‘ (gArByte[IdxPWMDuty] + u)*

The *actual duty cycle* stored in the global array , *gArByte[IdxPWMDuty]*‘ is defined as global array in the main program.

Within the *P control* function, the followind are declared as local variables

- *setpoint = gArLong[idxSetPM];*
- *measured = gArLong[idxFilter]; (Filtered output value is used as a measured)*
- Other local variables are shown within the function.

If the *setpoint* is specified in voltage, this should be scaled to the corresponding analog values using the scaling given in *Figure 10-7*. The scaling given in the figure is only valid for the current measurement electronic used for this 60Hz machine, for a different measurement electronic, new scaling for the measurement electronic is required to determine its range.

*Output of the Controller**:***

- The controller returns the duty cycle ‚*gArByte[IdxPWMDuty] + u*‘ to the function call *‚ChangeDuty()‘* as an input argument to write the value of duty cycle to the PWM output pin (*PwmOutPin 3*).

10.5.4.2 THE PROGRAM FLOW P-CONTROLLER FUNCTION

The program flow for the function is shown in *Figure 10-8*. The program code for the P controller is shown in *Appendix 11*.

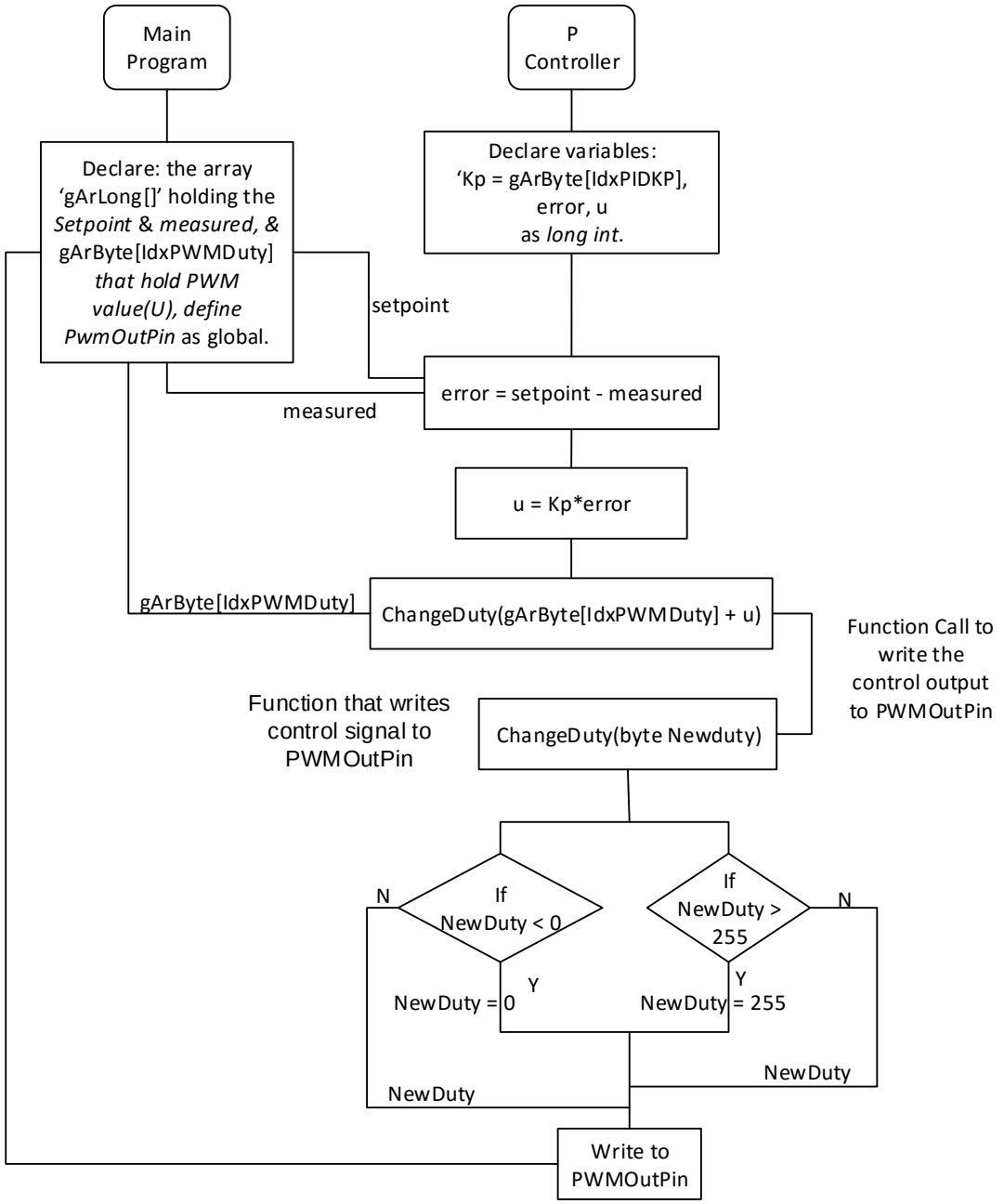


Figure 10-8: The Program Flow for the P Controller Function.

10.5.4.3 TESTING THE P CONTROLLER

The test in this section was carried out before implementing the controller on the 60Hz machine. *Figure 10-9* show a simple circuit for testing the P controller. In the figure the switch S introduces a disturbance (Load: Potentiometer) to the system when pressed. The controller should be able to return the output voltage to the setpoint once the load is withing the system capacity (4.5V).

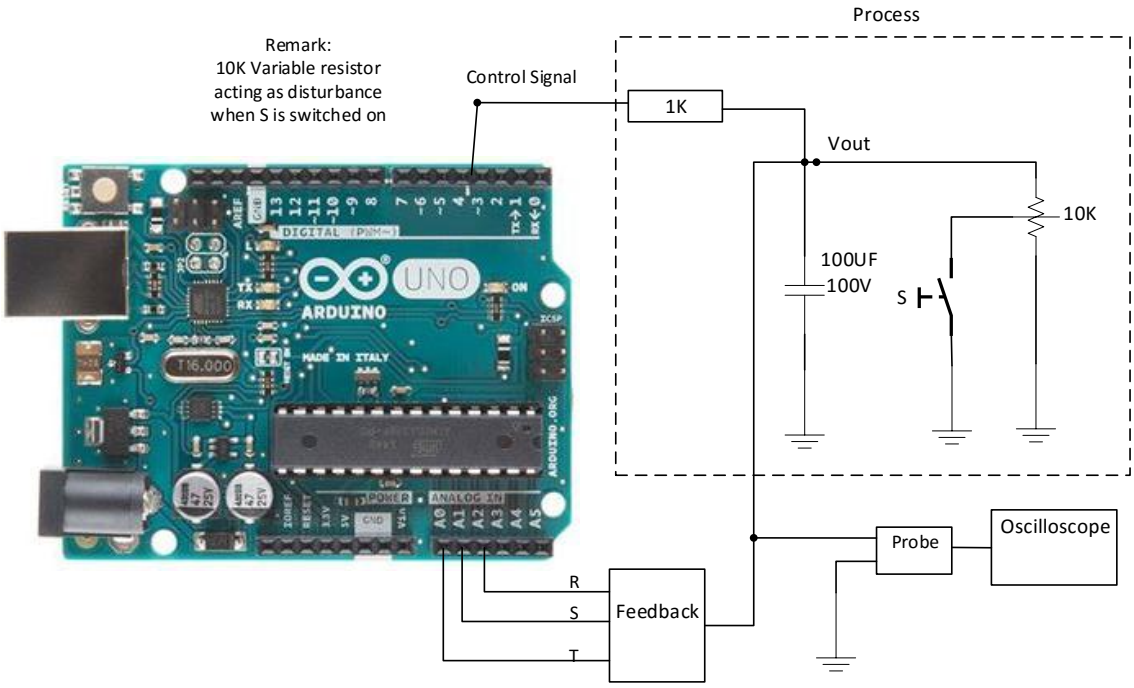


Figure 10-9: Hardware Connections for Testing the P Controller.

The output from the oscilloscope when the switch S is pressed for a setpoint changed from *1.0 volt* to *4.5volts* is shown in *Figure 10-10***.** The introduction of a disturbance causes the overshoot observed and the controller returns the output voltage to the setpoint.

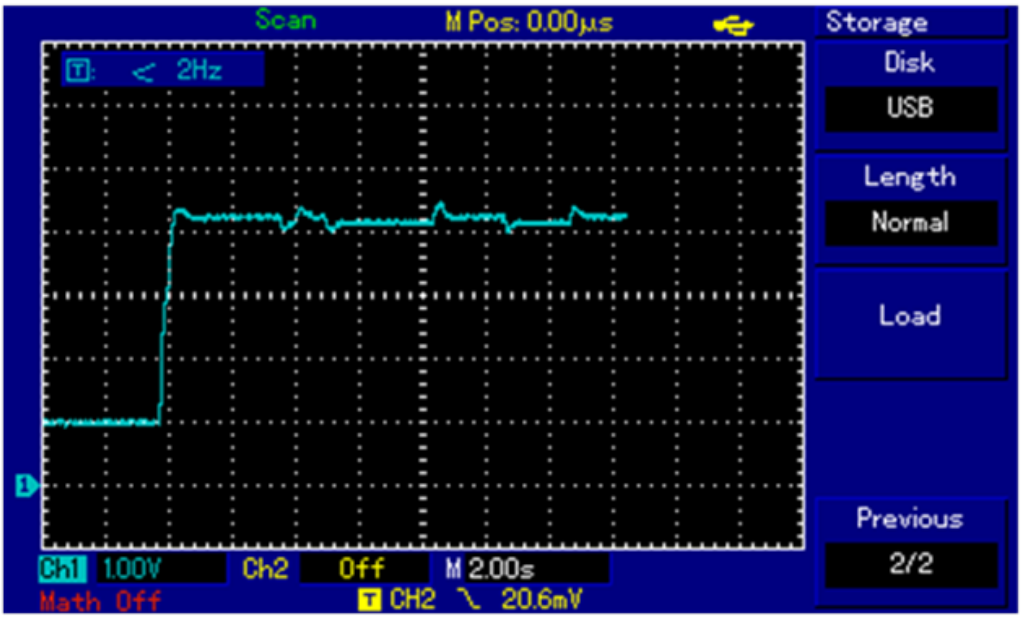


Figure 10-10: Testing the P Controller Before Implementing on the 60Hz Machine.

10.5.5 PI-CONTROL

The PI Control in this project is implemented as a function which can be inserted into any other project based on the guidelines stipulated in this section.

10.5.5.1 THE STRUCTURE OF THE PI CONTROL

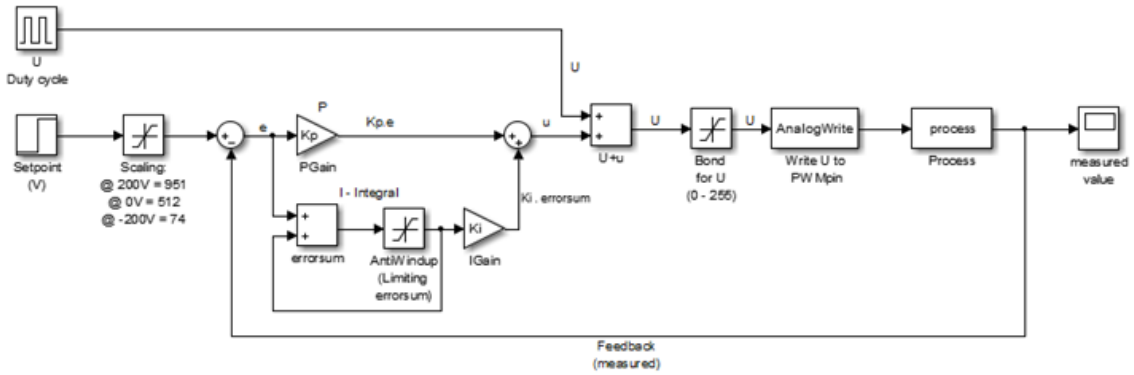


Figure 10-11: The Structure of the PI Controller Function.

Figure 10-11 shows the details setup in the PI controller function developed in this project. As shown in the figure, the measured output is compared with the setpoint and the difference is feed to the P (proportional) and the I (Integral) parts of the controller. The I part, add the previous difference (i.e. error) to the current error and a bond is set on the summed errors by implementing Antiwindup to avoid saturation of the controller signal. The error and the summed errors are respectively multiplied with proportional gain *Kp* and the integral time constant *Ki* and both outputs summed as control signal *u*. The control signal *u* is added to the actual duty cycle *U* and a function *‚ChangeDuty()‘* is called to writing the control output ‚*U*‘ to PWMOutPin 3 in order to vary the output voltage of the 60Hz machine (*process*). Within the function *‚ChangeDuty()‘* a bond is set on *U* between 0 and 255 to ensure the control output is within the PWM range *(0 - 255)*. That is preventing the saturation of the control signal.

The setpoint specifed in volatage can be scaled to the corresponding analog value as shown in the figure. This ensure that the voltage setpoint is converted to the corresponding analog value before it can be used to compare with the measured value in the controller function. The scaling shown is only valid for the current measurement

electronic used for this 60Hz machine, for a different measurement electronic, new scaling for the measurement electronic is needed to determine its range.

The prerequisites for integrating this function into another project are stipulated in this section and should be adhered to for the function to work.

Input to the controller function:

- *gArLong[idxSetPM]*
- *gArLong[idxFilter]*
- *Error sum defined stored in the as: gArLong[IdxErrorSum]*
- *PWM (Actual duty cycle) defined as: gArByte[IdxPWMDuty]*
- *Afunction call ‚ChangeDuty()‘ with an input argument: ‚(gArByte[IdxPWMDuty] + u)‘*

In the main program the following are declare as global variables array for storing the values shown:

- *Int errorsum: gArLong[IdxErrorSum]*
- *byte U for duty cycle: gArByte[IdxPWMDuty]*

Within the PI control function, declare the following as local variables:

- *long int setpoint = gArLong[idxSetPM];*
- *long int measured = gArLong[idxFilter];*
- *error = gArLong[idxSetPM] - gArLong[idxFilter];*
- other local variables are indicated within the function.

Output of the PI control:

- The controller returns the duty cycle ‚*gArByte[IdxPWMDuty] + u*‘ to the function call *‚ChangeDuty()‘* as an input argument to write the value of duty cycle to the PWM output pin (*PwmOutPin 3*).

The program code for the PI controller implemented is shown in *Appendix 11*.

10.5.5.2 THE PROGRAM FLOW FOR THE PI CONTROL FUNCTION

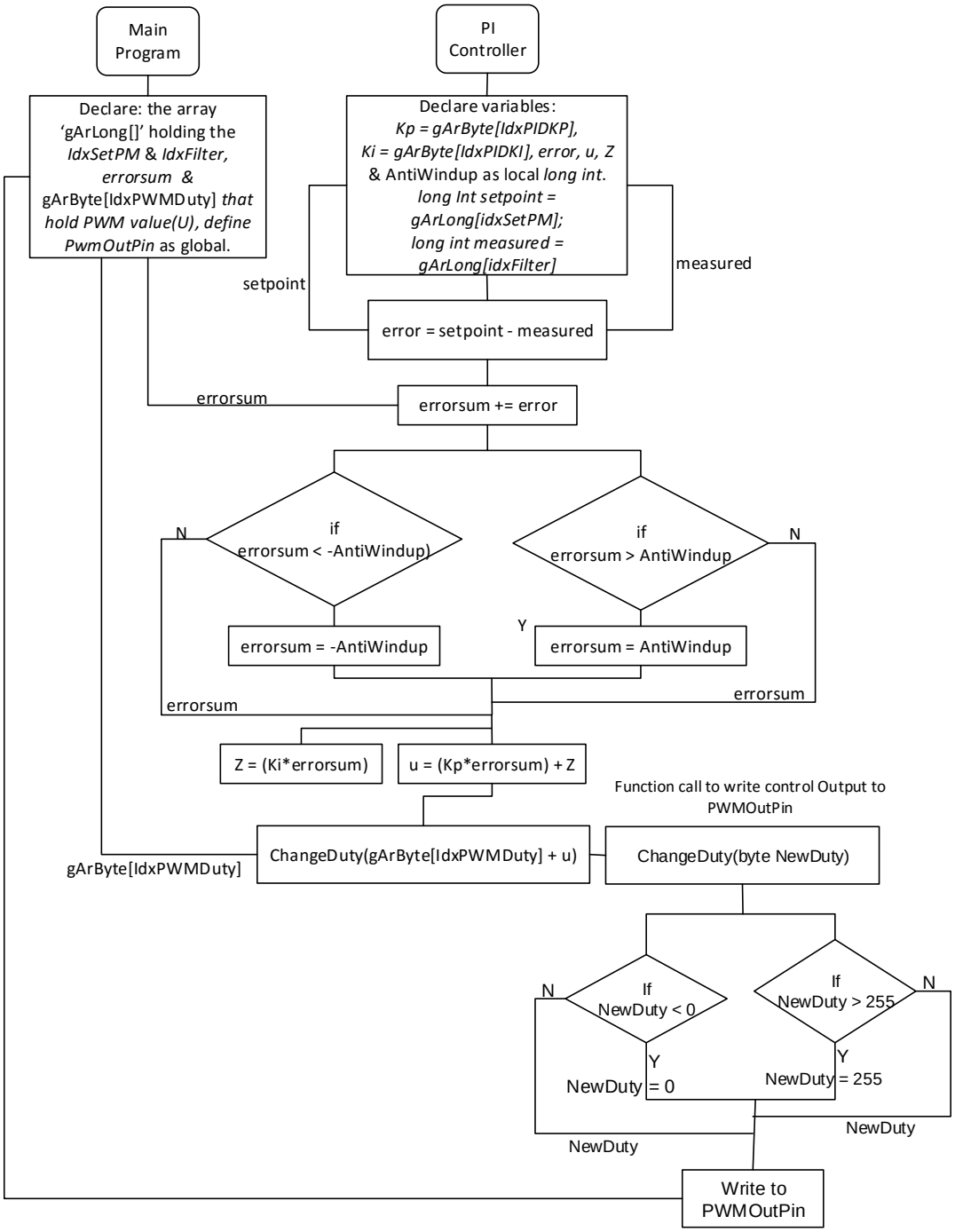


Figure 10-12: Program Flow for the PI Controller Function Implemented In this Project.

The program flow for the function is shown in *Figure 10-12*. This shows how the function is implemented.

10.5.5.3 TESTING THE PI CONTROLLER

The test in this section was carried out before implementing the controller on the 60Hz machine. *Figure 10-13* show a simple circuit for testing the *PI controller*. In the figure the switch S introduces a disturbance (Load: Potentiometer) to the system when pressed. The controller should be able to return the output voltage to the setpoint once the load is withing the system capacity (4.5V).

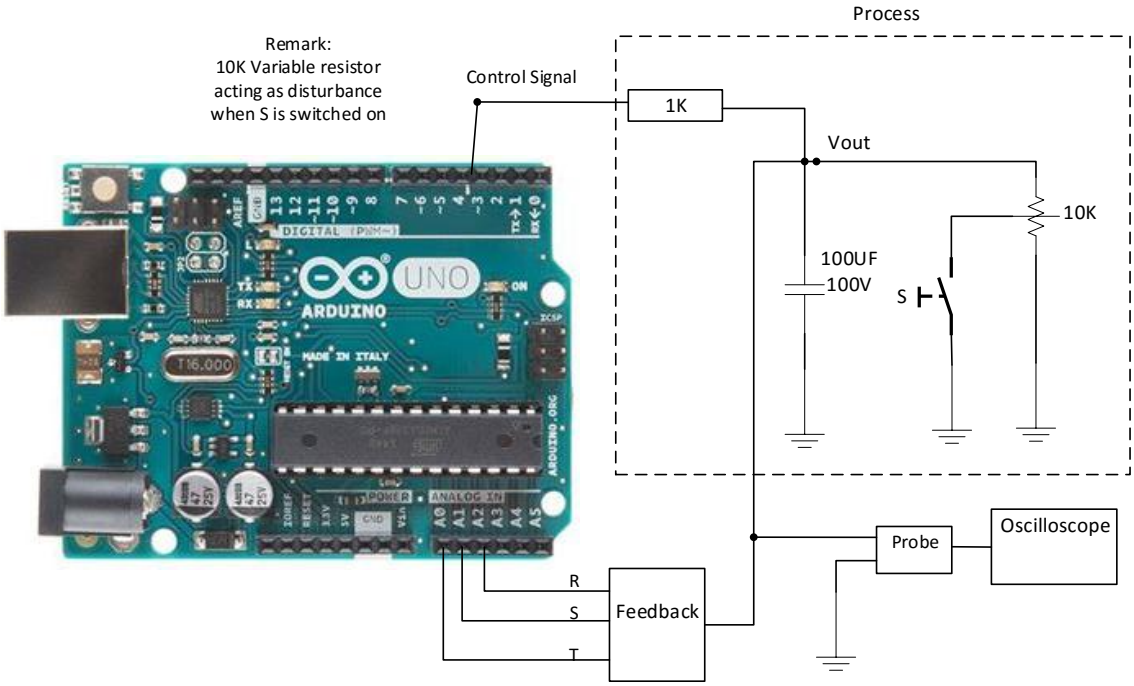


Figure 10-13: Hardware Connections for Testing the PI Controller.

The output from the oscilloscope when the switch S is pressed for a setpoint changed from 1 volt to 4.5 volts is shown in *Figure 10-14*. The introduction of a disturbance causes the overshoot observed and the controller returns the output voltage to the setpoint.

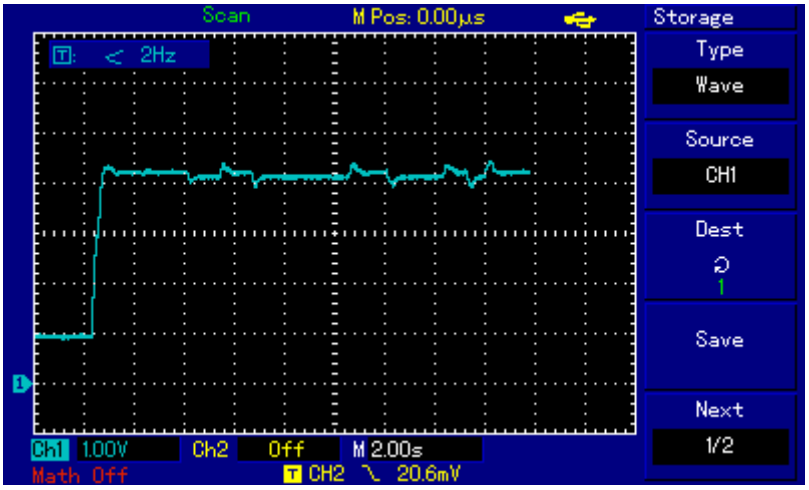


Figure 10-14: Testing the PI Controller Before Implementing on the 60Hz Machine.

10.5.6 PID CONTROL

The PID Control in this project is implemented as a function which can be inserted also into any other project based on the guidelines given in this section.

10.5.6.1 THE STRUCTURE OF THE PID CONTROL

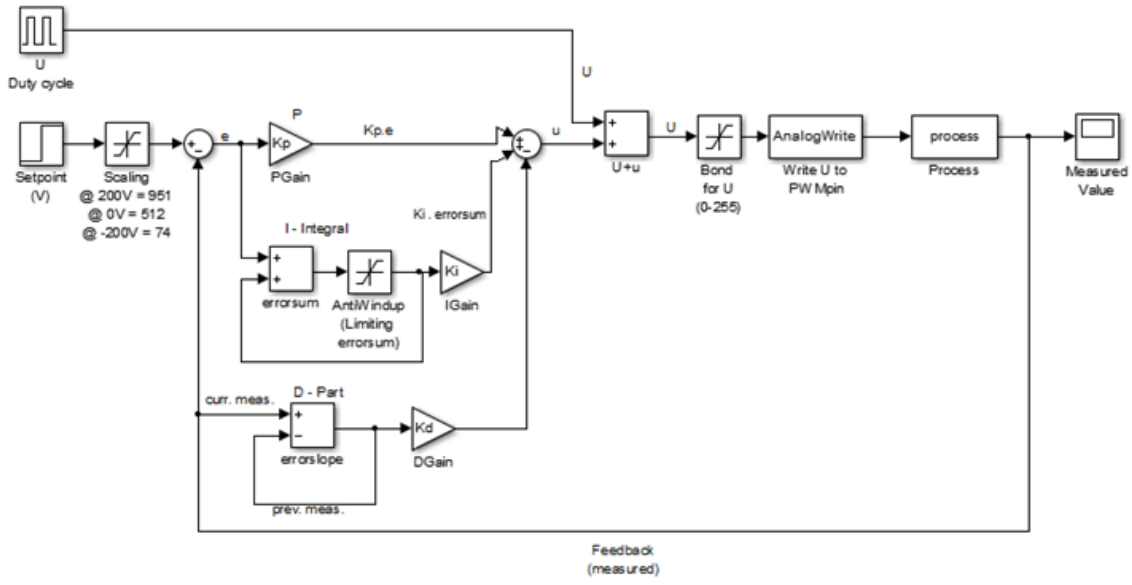


Figure 10-15: The Structure of the PID Control Implemented in this Project

Figure 10-15 shows the details setup of the PID controller function developed for this project. As shown in the figure, the measured output is compared with the setpoint and the difference is feed to the P (proportional) and the I (Integral) parts of the controller.

The D-part is determined by finding the slope for the measured value (*current measured - last measured*). The I part, add the previous difference (i.e. error) to the current error and set a limiter (bond) on the Integral (I-part) output (*errorsum*) by implementing Antiwindup to avoid saturation of the control signal. The outputs from P-part, I-part and D-part are respectively multiplied with their respective gain (*Kp, Ki, and Kd*) and summed up as control signal *u*.

The control signal *u* is added to the actual duty cycle *U* and a function *‚ChangeDuty()‘* is called to writing the control output ‚*U*‘ to *PWMOutPin 3* in order to vary the current of the excitation winding of the 60Hz machine (*process*) connected to the pin via excitation electronic board. Within the function *‚ChangeDuty()‘* a bond is set on the control output between 0 and 255 to ensure the control output is not saturated (i.e.: $0 \leq pwm \leq 255$) .

The setpoint specifed in volatage can be scaled to the corresponding analog value shown in the figure. This ensure that the voltage setpoint is converted to the corresponding analog value before it can be used to compare with the measured value in the controller. The scaling shown in the *Figure 10-15* is only valid for the current measurement electronic used for this 60Hz machine, for a different measurement electronic, new scaling for the measurement electronic would be needed to determine its range. The prerequisites for integrating this function into another project are given below and should be adhered to for the function to work.

Input to the controller function:

- *setpoint : gArLong[idxSetPM]*
- *measured : gArLong[idxFilter]*
- *Error sum defined stored in the array as: gArLong[IdxErrorSum]*
- *Current measure – Lastmeasure: gArLong[idxDFilter]*
- *PWM (Actual duty cycle) defined as: gArByte[IdxPWMDuty]*
- *Afunction call ‚ChangeDuty()‘ with an input argument: ‚(gArByte[IdxPWMDuty] + u)‘*

In the main program declare the following as global variables:

- *gArLong[idxSetPM]*

- *gArLong[idxFilter]*
- *gArLong[IdxErrorSum]*
- *gArLong[idxDFilter]*

Within the PID control function, declare the following as local variables:

- *long int setpoint = gArLong[idxSetPM];*
- *long int measured = gArLong[idxFilter];*
- other local variables are indicated within the function

Output of the PID control function:

- The controller returns the duty cycle ‚*gArByte[IdxPWMDuty] + u*‘ to the function call *‚ChangeDuty()‘* as an input argument to write the value of duty cycle to the PWM output pin (*PwmOutPin 3*).

10.5.6.2 THE PROGRAM FLOW FOR THE PID CONTROL FUNCTION

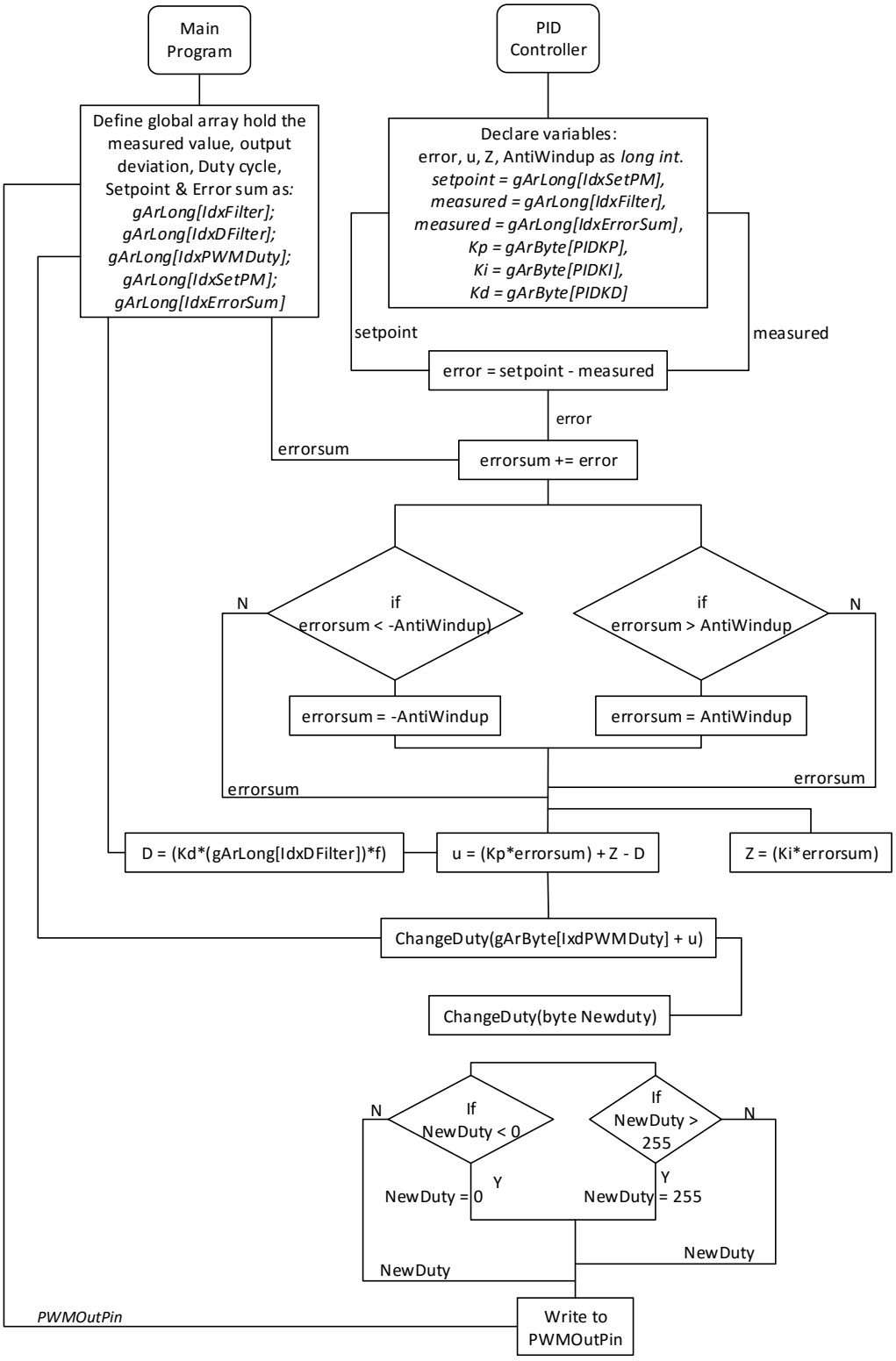


Figure 10-16: Program Flow for the PID Controller Function Implemented In this Project.

The program flow for the PID control is shown in *Figure 10-16*.

Remark: From the HMI, the PID controller settings (KP, KI and KD) determine which of the control strategies strategies (P, PI & PID) is being executed in the program. The settings are:

- *P - Control:* Is executed *when KI and KD are set to zero*
- *PI - Control:* Is executed *when and KD set to zero*
- *PID - Control:* Is executed *when non is set to zero.*

The code for the PID control is shown in *Appendix 11*.

10.5.6.3 TESTING THE PID CONTROL FUNCTION

The test in this section was carried out before implementing the PID controller on the 60Hz machine. *Figure 10-17* show a simple circuit for testing the *PID controller*. In the figure the switch S introduces a disturbance (Load: Potentiometer) to the system when pressed. The controller should be able to return the output voltage to the setpoint once the load is withing the system capacity (4.5V).

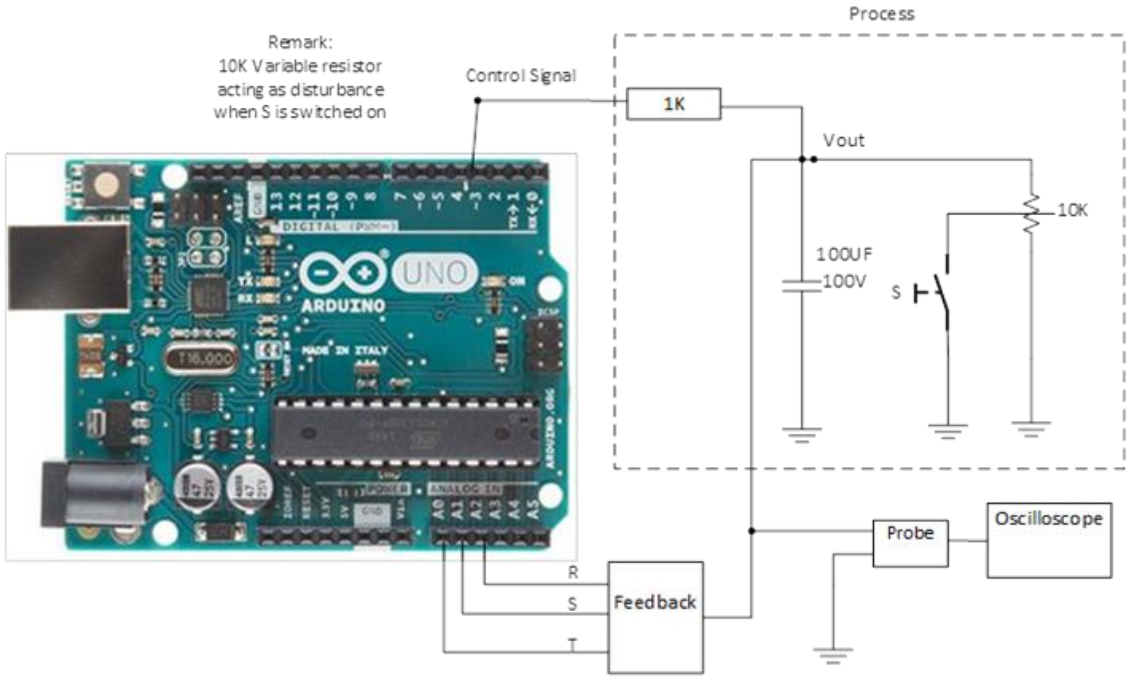


Figure 10-17: Hardware Connections for Testing the PID Controller.

The output from the oscilloscope when the switch S is pressed for a setpoint changed from 1 *volt* to 4.5 *volts* is shown in *Figure 10-18*. The introduction of a disturbance causes the overshoot observed and the controller returns the output voltage to the setpoint.

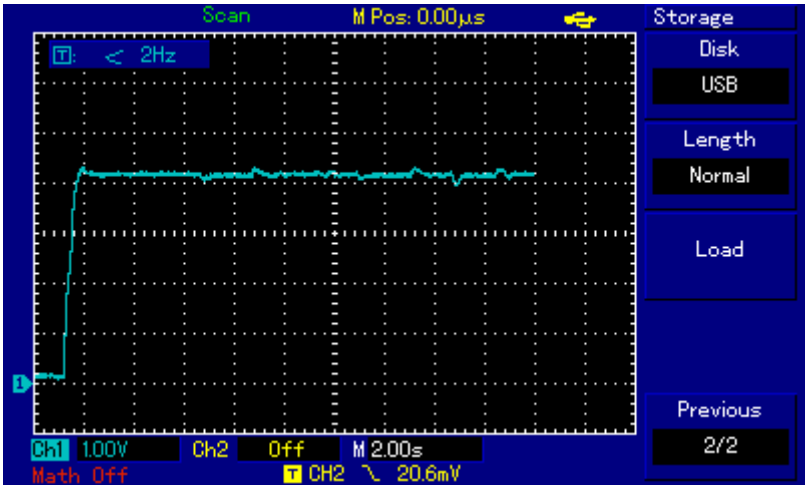


Figure 10-18: Testing the PID Controller Before Implementing on the 60Hz Machine

10.5.7 FUZZY LOGIC CONTROL

From the webserver interface (HMI), the field for the fuzzy logic setpoint is given a symbolic name ‚*gB5*‘ and its value is *2* (*i.e.: gB5=2*) where value 2 represnts fuzzy logic control. Once the setpoint for the fuzzy control is set, this changes the value in the global Array Byte gArByte[5] to 2 and this value represents *SymFuzzyControl* (in the main). This enable the fuzzy logic control function to be called in the *DoControl()* function to execute the function. The platform for this is shown in *Appendix 11*.

10.5.8 DOCONTROL FUNTION

This function switches between different control modes based on the control strategy specified by the user from the HMI (Web server). The different control strategies defined by this function are:

- *Open Loop Control*
- *PID Control (P, PI, PID)*
- *Fuzzy Logic Control*

- *Neural Network (NN)*

Each of this control functions or strategies is called and executed in this section based on the user interest. The structure of this function is shown in *Figure 10-19*.

The program code for the ‚*DoControl()*‘ function is shown in *Appendix 11*.

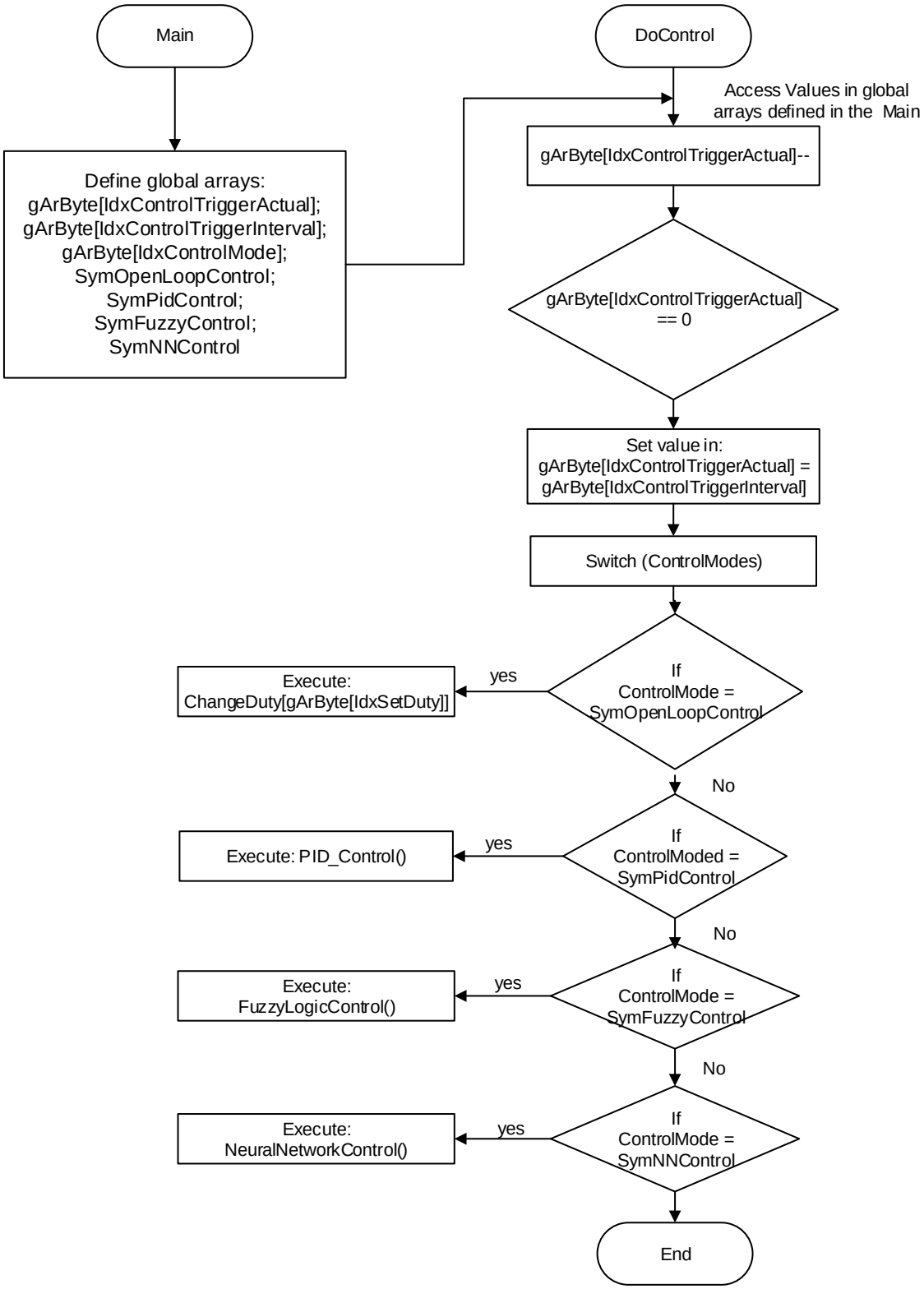


Figure 10-19: Program Flow for the 'DoControl()' Function

10.6 SCALING VOLTAGE SETPOINT TO ANALOG VALUE

Based on the scaling for the measurement electronic used for this project, the program code for calculating the voltage setpoint specified by the user (from HMI) to the corresponding analog values for the controller setpoint is developed as a function that can be inserted into any project based on the guidelines specified.

In the control function (P, PI, & PID)

- *Replace: error = gArLong[IdxSetMeasure] - gArLong[IdxFilter + gIdxNewest] with:*
  error = gArLong[IdxSetptVoltToSetPtMeasure]- gArLong[IdxFilter + gIdxNewest]

In the Main:

- Call the function: *ComputeVoltToSetpoint(gArLong[IdxSetMeasure])* with an input argument: *gArLong[IdxSetMeasure]* in the *void loop()* after the *DoWeb()* function.

In the HMI (User Interface):

- The convertion from voltage setpoint to the corresponding analog setpoint measure is only applicable for the close loop control setpoint.
- If this function is used a limit is set on the phase to phase voltage setpoint (0 - 253V) set by the user which is the maximum phase to phase voltage for the target machine (60Hz machine). The limit is set to avoid the user from given undesirable values which might lead to problem with the target machine. For a different machine, the limit should be set based on its maximum capacity (phase to phase output voltage).
- If the this funtion is not used to compute the corresponding analog value for a setpoint voltage, then the measured value of the RST can be obtained by setting the pwm for the open loop control and check the ‚System Status' under ‚Service' to get the measured value and then use this as the setpoint for the close loop control. The system status should be refreshed before reading the measured values or Filter value.

Caution:

Using this function involves the use of float to compute the actual measure for the setpoint and this requires more time and more process load than integer computaion.

The program code for this function is shown in *Appendix 8* with the code for the Main.

10.7 DATA LOGGING

This section logging the follwing data to an SD card.

- *Measurement of the three pahses: R, S, T*
- *Measured value: gArLong[idxFilter] for the three phases*
- *Control signal U*

The data logged to the SD card can be downloaded from the *Human Machine Interface* (HMI, i.e.: webserver).

The function is designed in such a way that the opening of the file for writing (logging) is implemented as a function: ‚*OpenWriteLogFile()*‘which is called in another function ‚*DoWeb()‘ (*under *Webserver*) to execute the code.

The decision to call the function ‚*OpenWriteLogFile()*‘ to open the file is based on the value in the global Array ‚*gArByte[idxCmd]‘* in *Table 10-1* and the decision to write to the file is based on value of the semaphore in the global Array ‚*gArByte[idxlogSemaphore]‘* in *Table 10-2*. The value in the global array *gArByte[idxlogSemaphore]* is used for timing control for writing to file (*logging*).

The global arrays ‚*gArByte[idxCmd]*‘ and ‚*gArByte[idxlogSemaphore]‘* can assume the values in *Table 10-1* and *Table 10-2* with their corresponding definitions.

Table 10-1: Different Contents of the gArByte[idxCmd] with their Corresponding definitions and Description

Content of *gArByte[idxCmd]*	Meaning of contents defined in the main program	Description
gArByte[idxCmd] = 0	SymCmdNone	No command is available
gArByte[idxCmd] = 1	SymCmdSetControl	
gArByte[idxCmd] = 2	SymCmdStartLog	Start data logging
gArByte[idxCmd] = 3	SymCmdStopLog	Stop data logging
gArByte[idxCmd] = 4	SymCmdStoreDefault	Store global variables as default to the SD card

Table 10-2: Different Contents of the gArByte[idxlogSemaphore] with their Corresponding definitions and Description

Content of *gArByte[idxlogSemaphore]*	Meaning of contents defined in the main program	Description
gArByte[idxlogSemaphore] = 0	SymLogOff	No Logging (i.e.: No Writing)
gArByte[idxlogSemaphore] = 1	SymLogWriteProgress	Writing to File in progress
gArByte[idxlogSemaphore] = 2	SymLogGetFirst	Reset the time axis *(numbering*) for datalogging
gArByte[idxlogSemaphore] = 3	SymLogGetNext	Update the Buffer for data logging

The sequence of operation is shown:

- The different values in the global Array: *gArByte[idxCmd]* is supplied by the user from the HMI (Human Machine Interface) which would be discussed later.
- When the value in: *gArByte[idxCmd]* = Sym*CmdStartLog*

(i.e.: SymCmdStartLog is 2 defined in the main program) the function ‚*OpenWriteLogFile()*' is called to open the file for writing the data.

- Once the file is opened, within the function ‚*OpenWriteLogFile()*' the content of the global array ‚*gArByte[idxlogSemaphore]*' *is set to* ‚*SymLogGetFirst*' to reset the time axis (*numbering*) for datalogging:
  gArByte[idxlogSemaphore] = SymLogGetFirst
  Therefore, the content in this array space would be 2.
- In the *ADC_ISR(),* the content in the array ‚*gArByte[idxlogSemaphore]*' is checked, if the value stored in the array space for idxlogSemaphore is equal 2 (*SymLogGetFirst*) the *ADC_ISR()* is forced to reset the time axis for the datalogging to 0.
- Then the content in ‚*gArByte[idxlogSemaphore]*' is checked again in the *ADC_ISR(),* if the value stored in the array space , *gArByte[idxlogSemaphore]*' is greater than or equal to 2 (*SymLogGetFirst*) the buffer for the datalogging are updated with new values by the *ADC_ISR().*
- After the buffers are updated, the ADC_ISR() resets the value stored in the array ‚*gArByte[idxlogSemaphore]*' to 1 (i.e. *SymLogWriteProgress*):
  gArByte[idxlogSemaphore] = SymLogWriteProgress
- After then, the content in the array ‚*gArByte[idxlogSemaphore]*' is checked by the main loop (*void loop()*) to see if the value is equal *1* (i.e. *SymLogWriteProgress*). If this is true, the data in the buffers are written to the SD card and the value in the array *gArByte[idxlogSemaphore]* is reset to *SymLogGetNext (i.e. 3)* for the *ADC_ISR()* function to updates the buffers.
- When the user clicked on ‚*Stop Logging*' from the *HMI* the value stored in the array ‚*gArByte[idxCmd]*' is reset to *3* (i.e. *SymCmdStopLog*) and this call a function: ‚*CloseLogFile()*' to closed the file.
- Finally when the file is closed, the value stored in the array ‚*gArByte[idxCmd]*' is reset to 0 (i.e. *SymCmdNone*).

The flow sequence for the data logging is shown in *Figure 10-20*.

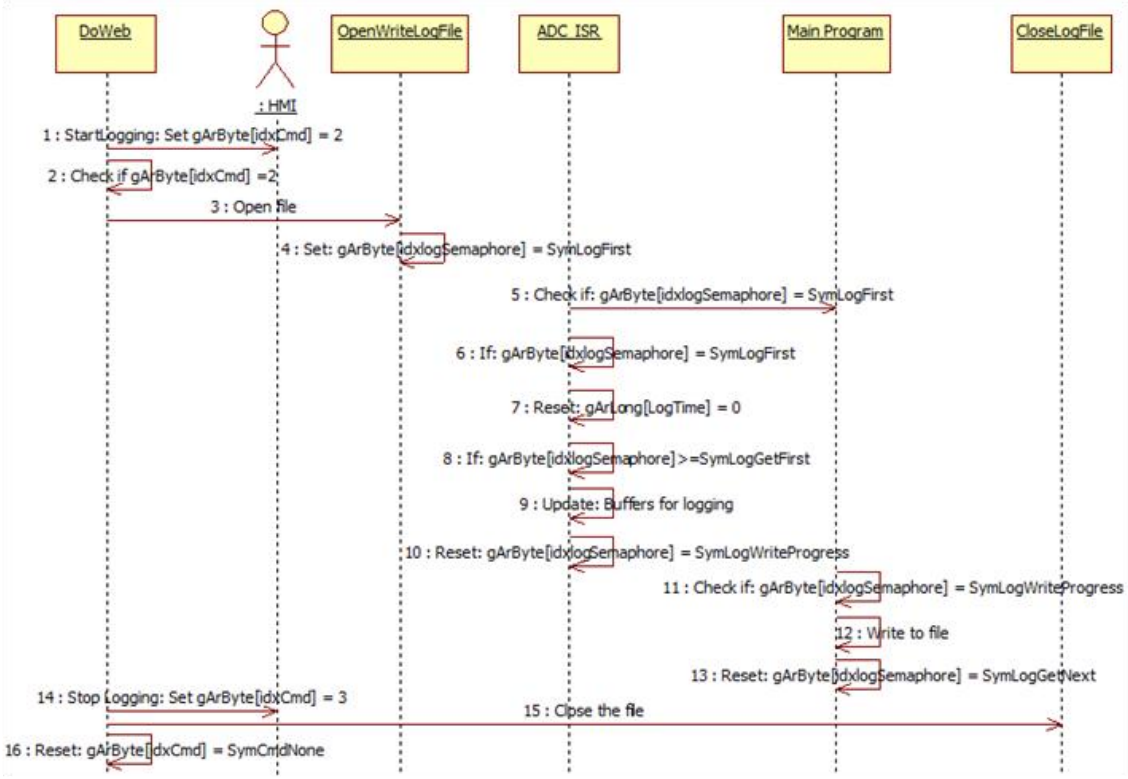


Figure 10-20: Data Logging Program Flow Sequence.

The program code for the ‚*OpenWriteLogFile()*‘ function is shown in *Appendix 12*.

The program flow sequence for the *CloseLogFile()* function is shown in *Figure 10-21*.

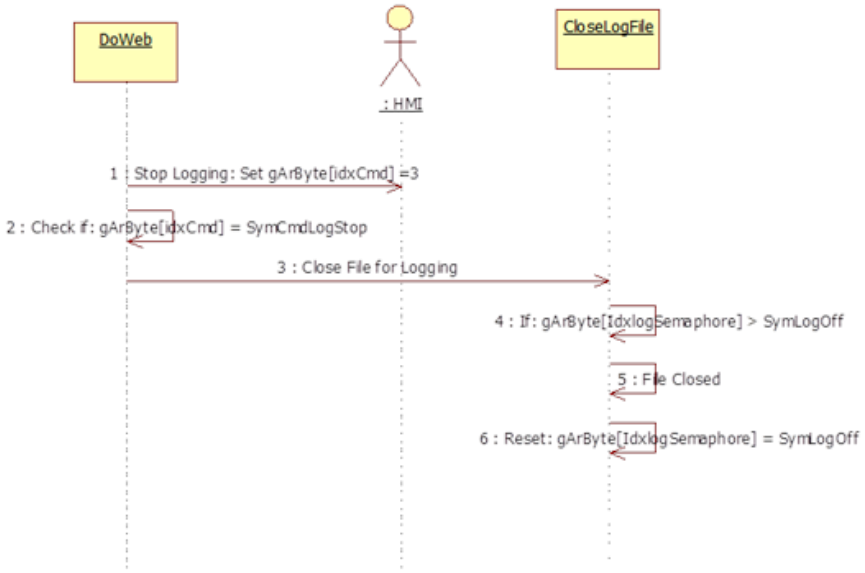


Figure 10-21: Program Flow Sequence for the CloseLogFile Function

The program code for the ‚*CloseLogFile()*‘ is also shown in *Appendix 12*.

10.8 WEBSERVER (HUMAN MACHINE INTERFACE)

The webserver implemented in this project was designed using Microsoft Expression Web. The welcome page and its service menu are shown in *Figure 10-22*.

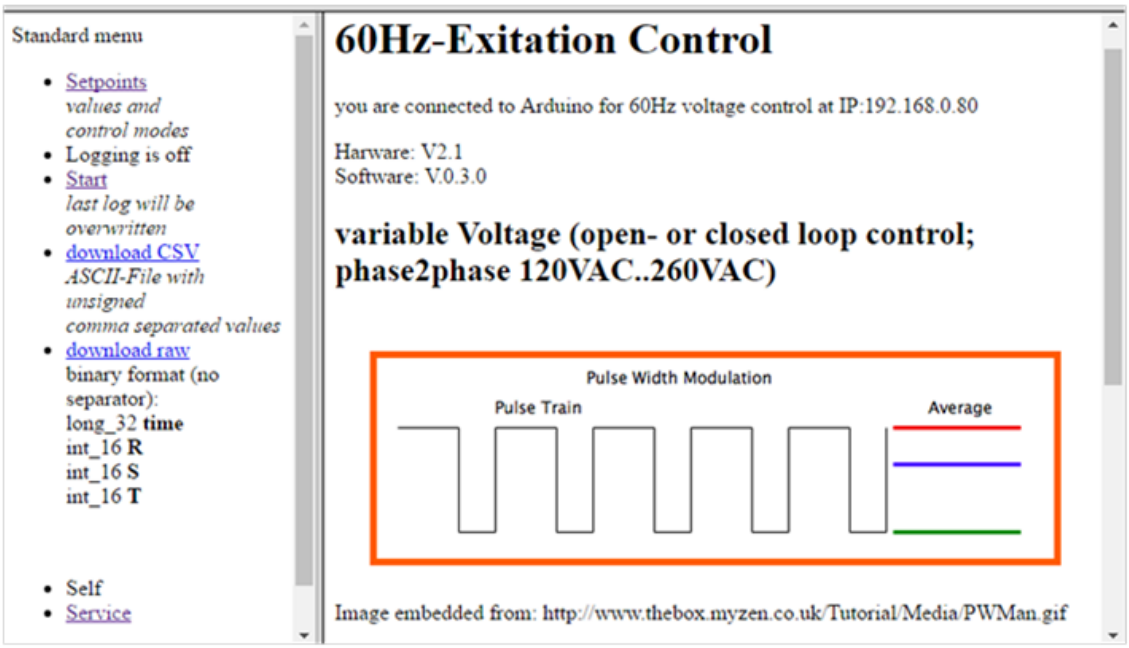


Figure 10-22: The Webserver Welcome Page and its Service Menu.

10.8.1 WEBSITE LOGIC

The logic design for the webserver (*HMI*) is describe in shown in *Figure 10-23*.

Each field in *Figure 10-23* (webserver) is represented with a *symbolic name* with either:

- *gBIdx: where Idx = 1, 2, ..., n, eg.: gB0, gB1,..., gBn.*
- *gIIdx: where Idx = 1, 2, ..., n, eg.: gI0, gI1, ..., gIn. or*
- *gLIdx: where Idx = 1, 2, ..., n, eg.: gL0, gL1, ..., gLn.*

where:

- *gB* – means global Byte whose values are stored in the global Array Byte (*gArByte[]*) in the array space ‚*Idx*‘ value. The global Array Byte is defined in the Main program.

- *gI* – means global Interger whose values are stored in the global Array Interger (*gArINT[]*) in the array space *‚Idx‘* value (*0, 1, 2, ..., n*). The global Array Interger is defined in the Main program.
- *gL* – means global Long Interger whose values are stored in the global Array Long (gArLong[]) in the array space *‚Idx‘* value. The global Array Long is defined in the Main program.

n – represents the numbers for each *symbolic name* (*gB, gI and gL*) in the webserver.

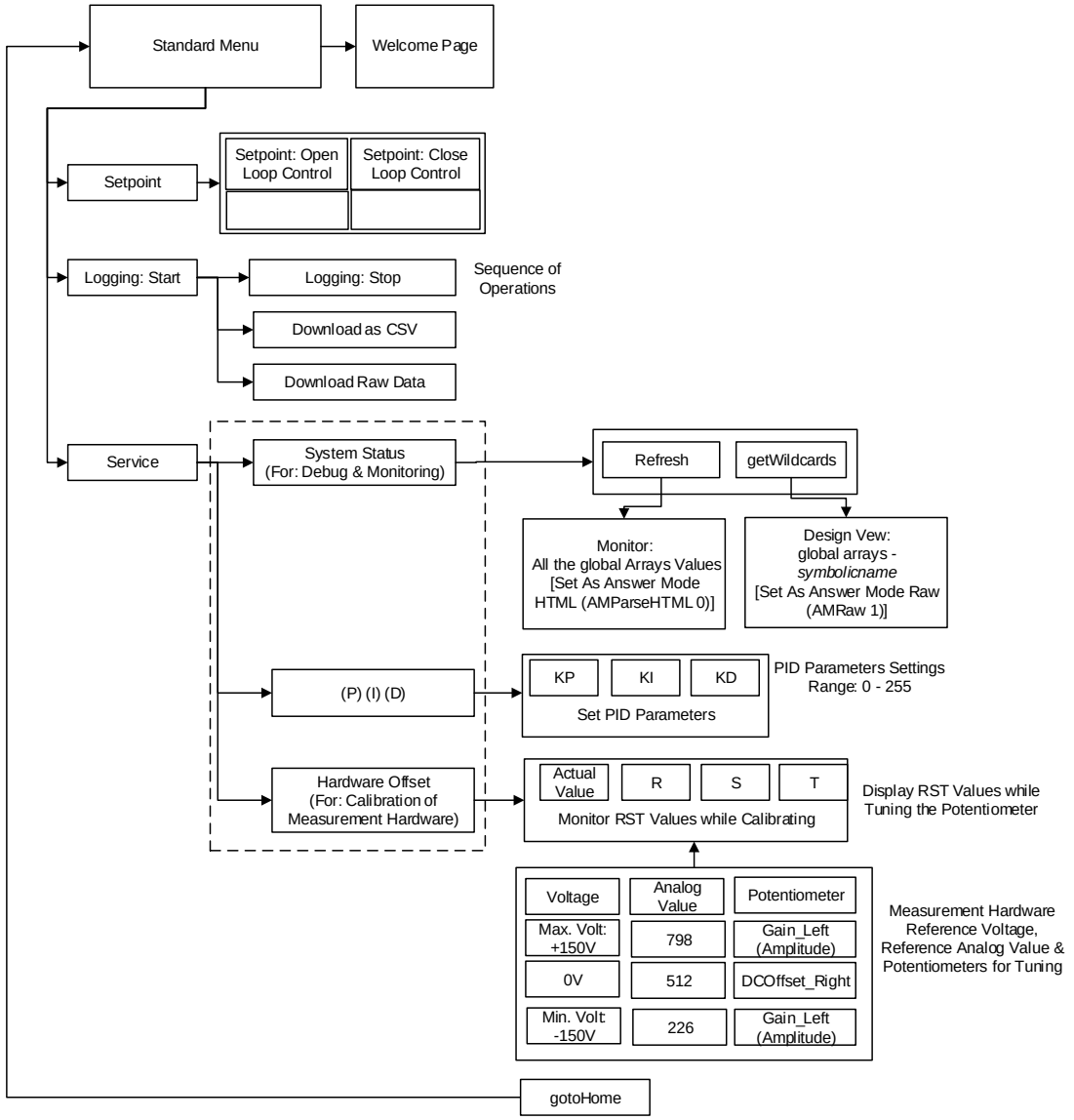


Figure 10-23: Logic Design for the Webserver (HMI).

Each values for each *symbolic name* (gB, gI& gL) are respectively stored in each of their global array (defined in the main program) in a space defined by their *Idx* number.

Once the user click any of the fields in the HMI, the *symbolic name* and/or *symbolic name with the value* (*symbolic name=value, eg.: gB0=1*) for that field is sent with the request string from the web (client) to the server (Arduino) to processing the request and respond to the request. The design view for the *symbolic names* for the global Byte (gBIdx), global Interger (gIIdx) and the global Long (gLIdx) are shown in *Figure 10-24*, *Figure 10-25* and *Figure 10-26* respectively.

Global Byte Array

Array index	Meaning of Index	place holder	Meaning of content of volatile byte gArByte[11]:
0	logSemaphore 0	#gB0#	LogOff 0; WriteProgress 1; GetFirst 2; GetNext 3
1	ADCRun 1	#gB1#	Channel just scanned
2	**WebAnswerMode**	#gB2#	**AMRaw 0; AMParsedHTML 1; AMCSV 2**
3	IdxSetDuty	#gB3#	Setpoint for open loop control
4	IdxControlMode	#gB4#	OpenLoopControl 0; PidControl 1; FuzzyControl 2; NN 3; Test 4
5	PIDKP	#gB5#	parameter for PID-Control
6	PIDKI	#gB6#	parameter for PID-Control
7	PIDKD	#gB7#	parameter for PID-Control
8	PWMDuty	#gB8#	backuped actual duty cycle (at PwmOutPin 3)
9	**IdxCmd**	#gB9#	**CmdNone 0; CmdSetControl 1; CmdStartLog 2; CmdStopLog 3 (set to 0 after execution)**
10	ControlTrigger actual value	#gB10#	count down for change of Duty cycle in control
11	ControlTrigger interval	#gB11#	sequence of Duty cycle in control (start value of count down)
12	byte spy	#gB12#	option to inspect or change any by value by inserting a simple command in the source code for debug purpose.

Figure 10-24: Design for the Symbolic Name Representation of the Global Byte (gBIdx) Implemented in the Webserver

The ‚*<#symbolicName#>*‘ (eg.: *#gB#*) are referred to *place holders (values)* and these are replaced with the results of the request sent by webserver to the client (web). The user can also place a ‚value‘ and send it to the webserver as request. Hence the communication is bi-directional. Once the request is made, all the fields will display the current settings in each global arrays for the user to monitor and see what is happening in the program. The *symbolic name* (eg.: gB0, gB1 gB2, etc) are the attribute names given given to each field and this are passed in the request string from the client (web) to the webserver to analyze and respond to the request. The same procedures applies to *Figure 10-25* and *Figure 10-26*.

Global Integer Array

Array index	Meaning of Index	place holder	Meaning of content of volatile int gArInt[ngINT];
0	R	#gI0#	sampled Value for phase R directly
1	S	#gI1#	sampled Value for phase S directly
2	T	#gI2#	sampled Value for phase T directly
3	BufR	#gI3#	buffered to have it consistent
4	BufS	#gI4#	buffered to have it consistent
5	BufT	#gI5#	buffered to have it consistent

Figure 10-25: Design for the Symbolic Name Representation of the Global Interger (gIIdx) Implemented in the Webserver.

Global Long Integer Array

Array index	Meaning of Index	place holder	Meaning of content
0	Measure	#gL0#	newest or previous (toggeled)
1	Measure	#gL1#	newest or previous (toggeled)
2	Filter	#gL2#	newest or previous (toggeled)
3	Filter	#gL3#	newest or previous (toggeled)
4	DMeasure	#gL4#	derivative
5	DFilter	#gL5#	derivative
6	IdxSetPM	#gL6#	Setpoint as unit measure
7	DCOffset0	#gL7#	computed Value for analog input as 0V
8	Idle	#gL8#	number of idle loops in main
9	LogTime	#gL9#	time axis for logged values (gaps might happen)
10	LogTimeBuf	#gL10#	consistent ot samples
11	SetMeasure	#gL11#	setpoint given as unit measure

Figure 10-26: Design for the Symbolic Name Representation of the Global Long (gLIdx) Implemented in the Webserver.

The prgram flow for the webserver is shown in *Figure 10-27*.

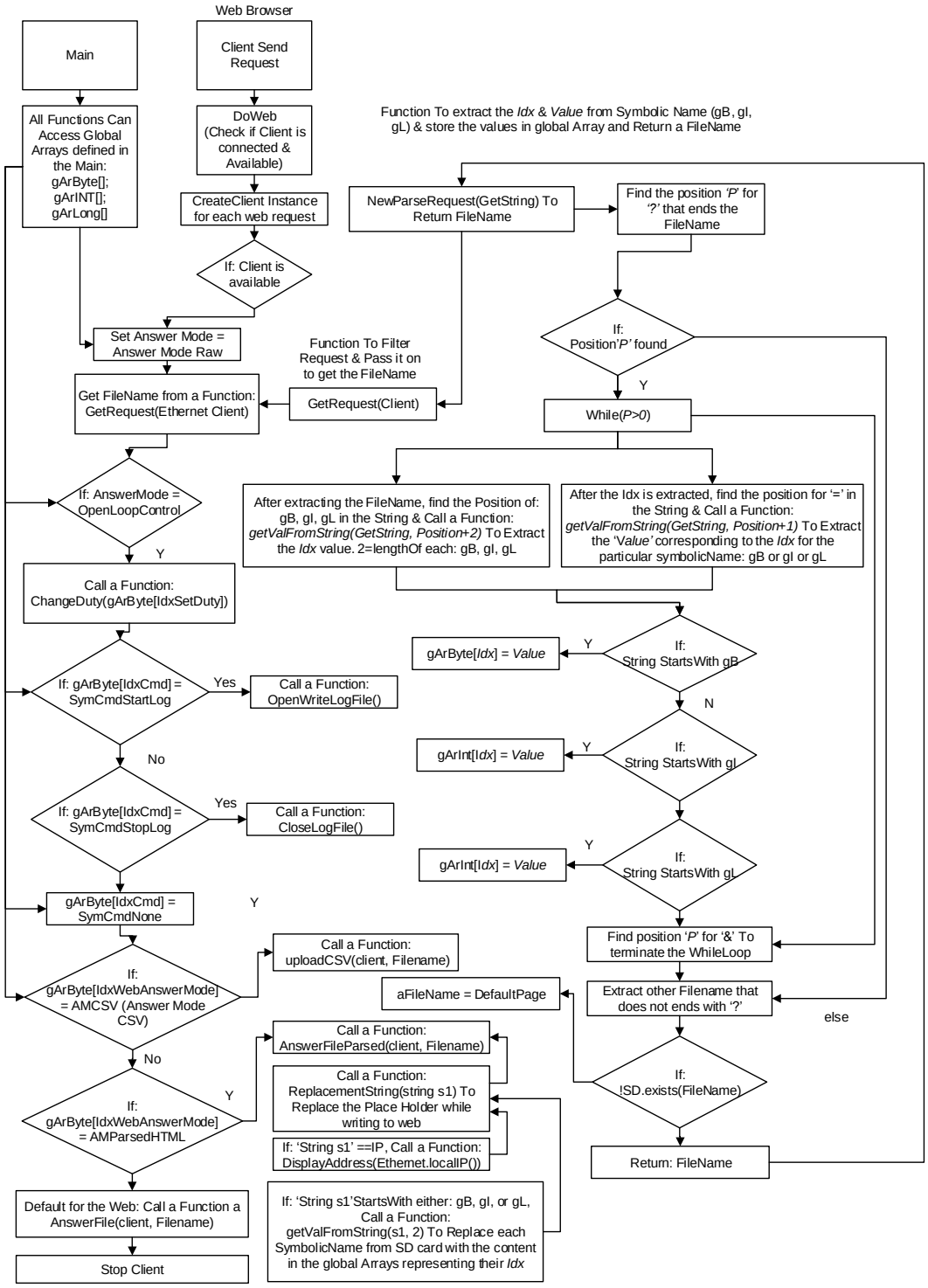


Figure 10-27: The Program Flow Sequence for the Webserver Implemented in this Project.

10.8.2 ANALYZE REQUEST

From *Table 10-3* each submenu in *Figure 10-22* is designed with a a speific file name and each of this has a corresponding filename stored in the SD card which are used to respond (‚*answer*‘) to web request from the client based on the file and field clicked by the user. *Table 10-3* (b) the file name is ‚Setpts.htm‘ with a field setpoint measure (Close loop setpoint). The attribute name for this field is ‚*gL11*‘ (*symbolic name*) and its *place holder* is ‚*#gL11#*‘ shown in (b). The user supply 230 as a request for the setpoint measure to the webserver (Arduino). The *file name*, *attribute name* (*symbolic name*) ‚*gL11*‘ and its *value* (*230*) are sent with the request string from the client (web browser) to the webserver as shown in the first row of *Figure 10-28*.

Table 10-3: Request Sent by the User from the Web to the Webserver

setpoint duty cycle (0:255) for open loop control	setpoint measure for close loop control
#gB3#	#gL11#
	Measure, take as setpoint
	#gL0#

(a) Filename: SetPts.htm

setpoint duty cycle (0:255) for open loop control	setpoint measure for close loop control
0	230
	Measure, take as setpoint
	-785695

(a) Filename: SetPts.htm

```
GET /SetPts.htm?gL11=230&gB2=1&gB4=1 HTTP/1.1
Host: 192.168.0.80
Connection: keep-alive
Accept: text/html,application/xhtml+xml,applicatio
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0) AppleWeb
Referer: http://192.168.0.80/SetPts.htm?gL11=230&g
Accept-Encoding: gzip, deflate, sdch
Accept-Language: en-US,en;q=0.8
```

Figure 10-28: Request String from the Client (Web Browser) to the Webserver (Arduino).

This request is analyzed and filtered by the function ‚*GetRequest(EthernetClient client)*' as shown.

```
GET /SetPts.htm?gL11=230&gB2=1&gB4=1 HTTP/1.1
GET /SetPts.htm?gB3=0&gB2=1&gB4=0 HTTP/1.1
```

The function , *NewParseRequest(String GetString)*' extracts the filename *(SetPts.htm),* the vlaue (*230*) for the gL11 and store this value (230) into the global array ‚gArLong[Idx]' with the Idx value ‚Idx=11' specified in the string (*gL11*) and return the filename to the function ‚*DoWeb()*'. The controller can access this values ‚*230*' in the global array long ‚*gArLong[11]*' and use this as a setpoint for the control. The same procedures applies to *gB2=1* and *gB4=1*. For the global Byte ‚*gB2=1*' and *gB4=1*, their values ‚1' are stored respectively in the global Array Byte ‚*gArByte[2]'* and *gArByte[4]*.

The values in the global Array Byte ‚*gArByte[2]*' is used for Web Answer Mode ‚*WebAnswerMode*'. When any of the close loop parameters (eg.: setpoint measure, KP, KD, & KD) are clicked by the user the value in *gArByte[4]* is set to 1, (i.e.: *gb4=1*) to indicates that the system is on close loop control mode and vise versal. This is an added feature to inform the user baout the control mode without having to navigate to another field to check the actual parameter responsible for this information.

The ‚*DoWeb()*' can call different functions responsible for the different answer mode based on the value written to the global array byte ‚*gArByte[2]*'. The codes for the functions discussed are shown in *Appendix 13* in the Webserver.

10.8.3 KINDS OF ANSWERING

There are three modes of answering the web client request

- *AMParsedHTML: defined as 0*
- *AMRaw: defined as 1*
- *AMCSV: defined as 2*

The global Array Byte ‚gArByte[2]' is responsible for this mode. If the value written in this array is 0 (i.e.: gB2=0) the function ‚*AnswerFileParsed(client, Filename)*' responsible for sending the reponse to the request in HTML is called. This function search for the filename given in the request string in the SD Card and upload the data to

the web with the information requested as HTML (for design view and debugging). The code for the function *AnswerFileParsed(client, Filename)*‘ is shown in *Appendix 13* with the webserver.

If the value written in this array is 1 (i.e.: gB2=1) the function ‚*AnswerFile(client, Filename)*‘ which is the default answer mode (AMRaw) is called. This function search for the filename given in the request string in the SD Card and upload the data to the web with the information requested. The code for the function *AnswerFile(client, Filename)* is shown in *Appendix 13* with the webserver.

Also if the value written in this array is 2 (i.e.: gB2=2) the function ‚*uploadCSV(client, Filename)*‘ which is the answer mode (AMCSV) is called. This function search for the filename given in the request string in the SD Card and upload the data to the web with the information requested in CSV format. The code for the function *uploadCSV(client, Filename)* is shown in *Appendix 13* with the webserver.

As the files are being uploaded to the client (web) from the SD Card, the place holders (*#symbolicname#*) for example *#gL11#* in the file for the HTML fields ‚*Attribute: value*‘ are replaced with the values in the global array corresponding to their array and Idx value.

- *gBIdx: Is replaced with value in the global Array Byte: gArByte[Idx]*
- *gIIdx: Is replaced with value in the global Array Byte: gArINT[Idx]*
- *gLIdx: is replaced with value in the global Array Byte: gArLong[Idx]*

Remark: All the codes for each function can be found in *Appendix 13* in the Webserver.

10.8.4 HTML- PAGES

The web pages for the welcome page, service, PID control (P, PI, & PID), System status, Hardware offset (Hardware calibration) are shown in the following figures.

The HTML codes are shown in *Appendix 14*. Each HTML codes are stored as a file with their name in *Appendix 14*. The various animation files and the HTML codes are also stored in the SD Card attached to back cover of this project.

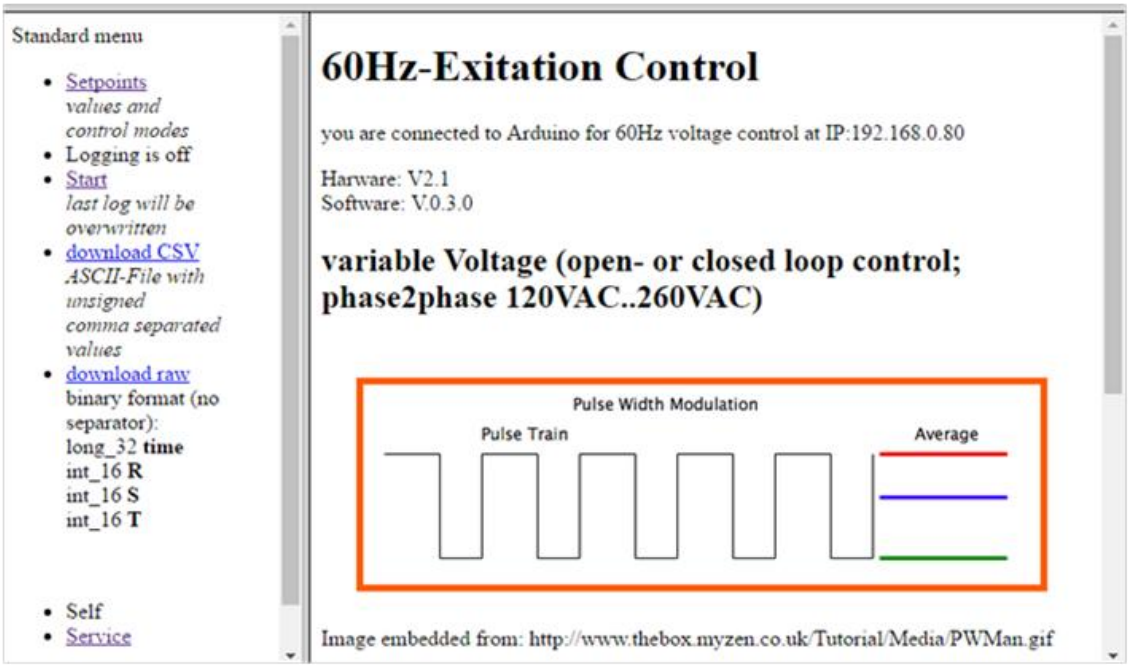


Figure 10-29: Welcome Page for the 60Hz Machine.

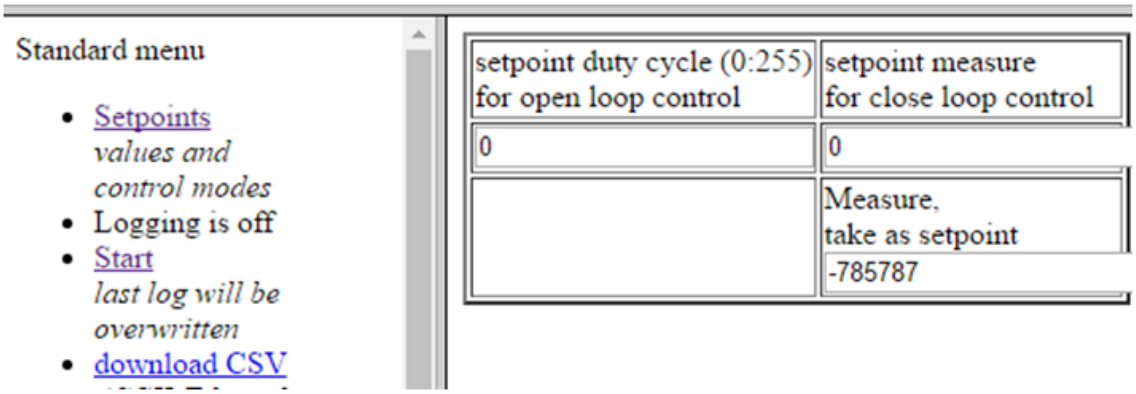


Figure 10-30: Page for the Open Loop and Close Loop Setpoint for the 60Hz Machine.

Figure 10-31: Data Logging Page for the 60Hz Machine.

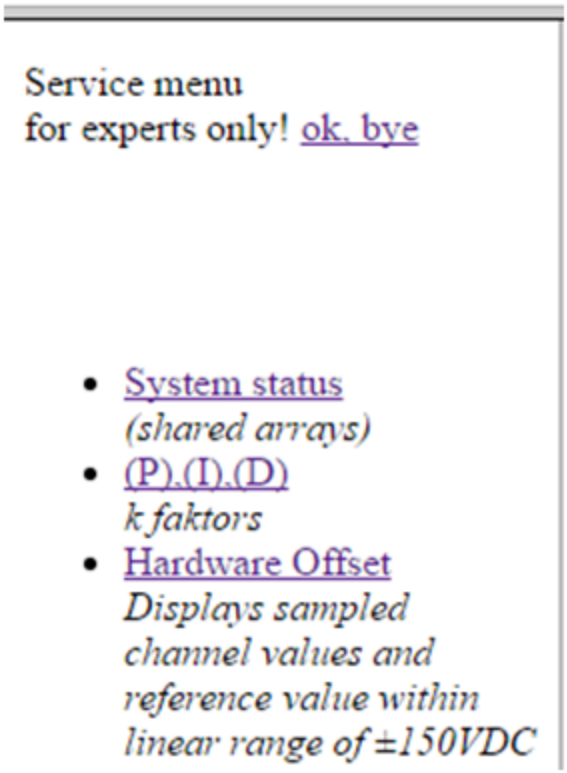
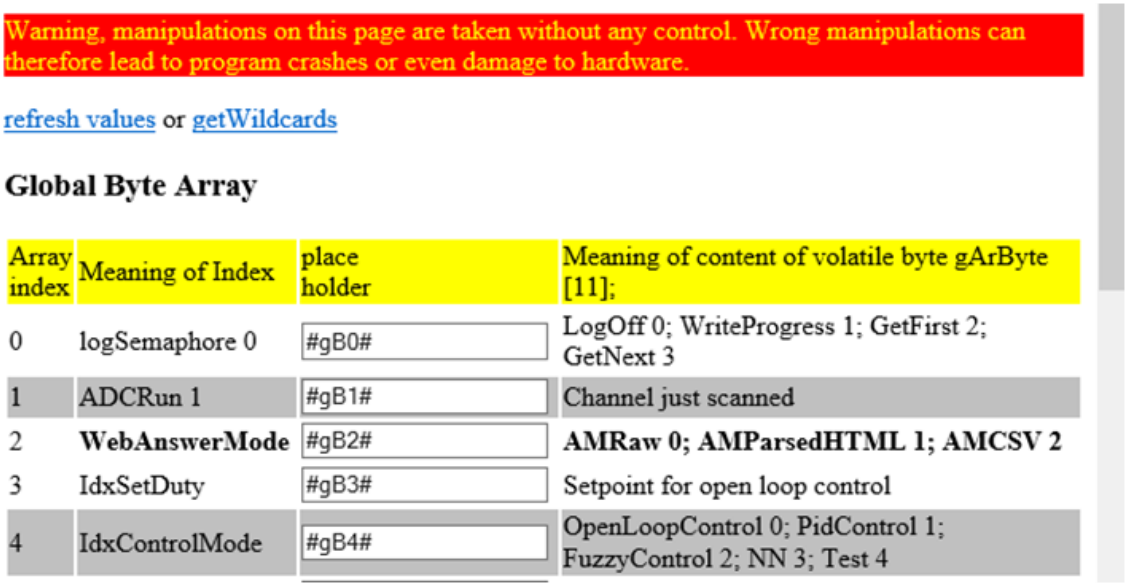


Figure 10-32: Service Page for the Control of 60Hz Machine.



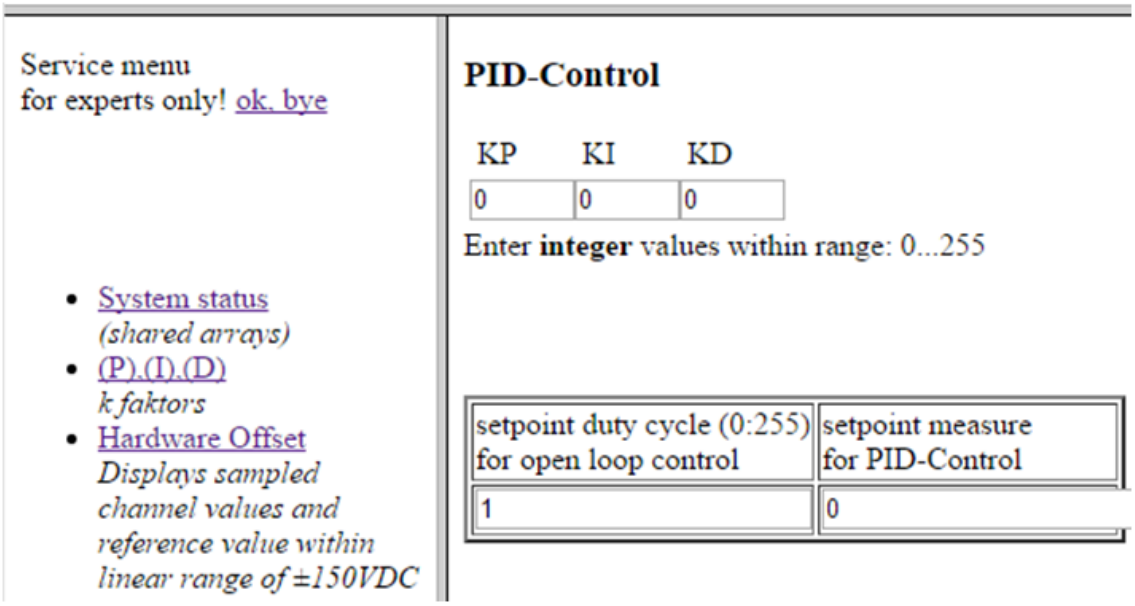
Warning, manipulations on this page are taken without any control. Wrong manipulations can therefore lead to program crashes or even damage to hardware.

refresh values or getWildcards

Global Byte Array

Array index	Meaning of Index	place holder	Meaning of content of volatile byte gArByte [11];
0	logSemaphore 0	#gB0#	LogOff 0; WriteProgress 1; GetFirst 2; GetNext 3
1	ADCRun 1	#gB1#	Channel just scanned
2	**WebAnswerMode**	#gB2#	**AMRaw 0; AMParsedHTML 1; AMCSV 2**
3	IdxSetDuty	#gB3#	Setpoint for open loop control
4	IdxControlMode	#gB4#	OpenLoopControl 0; PidControl 1; FuzzyControl 2; NN 3; Test 4

Figure 10-33: Service Status Page for the 60Hz Machine.



Service menu
for experts only! ok, bye

- System status
  (shared arrays)
- (P),(I),(D)
  k faktors
- Hardware Offset
  Displays sampled channel values and reference value within linear range of ±150VDC

PID-Control

KP	KI	KD
0	0	0

Enter **integer** values within range: 0...255

setpoint duty cycle (0:255) for open loop control	setpoint measure for PID-Control
1	0

Figure 10-34: PID Control (P, PI & PID) Page for the Control of the 60Hz Machine.

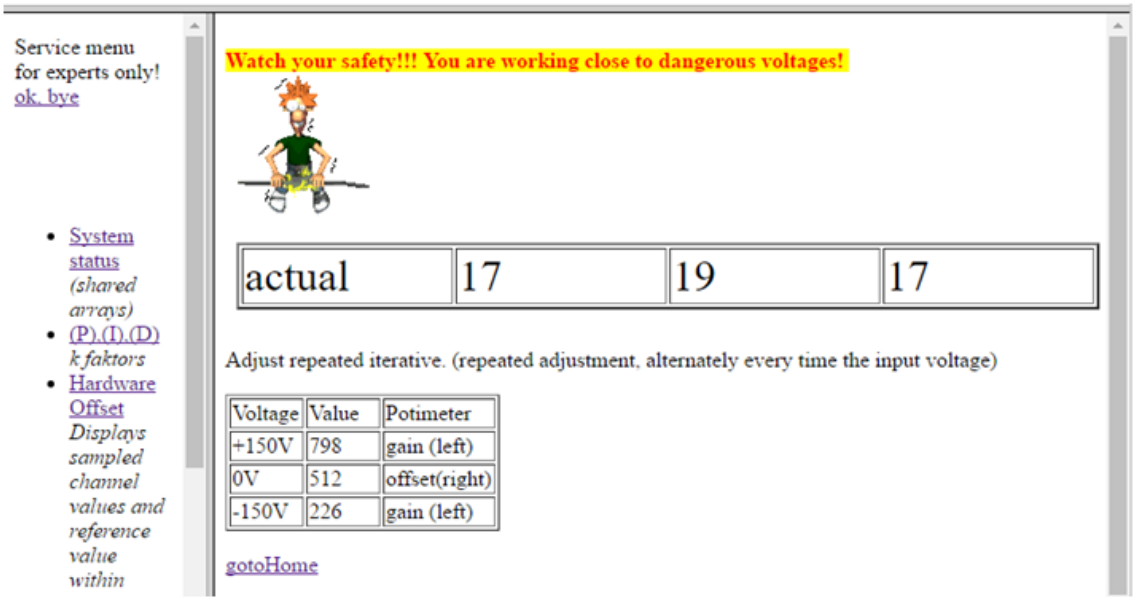


Figure 10-35: Hardware Offset Calibration Page for the 60Hz Machine Measureement Electronic.

10.8.5 HEPTIC IN MONITORING AND CONTROL

For the monitoring and control using the HMI interface, the following guidelines should be adhered.

For open loop control:

- Set the open loop setpoint. The setpoint shoould be given in duty cycle.
- Duty cycle of 54 (i.e.: pwm = 54) correspond to the phase-to-phase nominal voltage (*220V*) for the machine and a measured value of 194287.
- The duty cycle of 60 correspond to 230V phase-to-phase voltage and a measured value of 211000.
- For a phase to phase voltage of 120V, the measure is 200000.

For close loop control:

- Set the , *ControlTriggerInterva/* ‘ to a value greater than 2 for fast control.
- Set the *KP = 136*. (*KP = 128, 132 and 144 will also work*)
- If desired to operate in P Control, set *KI = KD = 0*
- If desired to operate in PI Control, set *KP = 136* and *KI = 131* and *KD = 0*.

- If desired to operate in PID Control, set *KP = 136, KI = 131* and *KD = 176.*
- Set the setpoint measure for the close loop to *194287* for the nominal voltage *220V.*
- In the void setup() in the main loop, duty cycle of 255 is written to the output pin of the connected to the machine to set the output voltage for the machine to zero. This is done due to the inversion of optical signal (control signal) by the opto isolator on the excitation electronic board.

Extending the Code:

- It should be noted that all the variables contents are store in three global arrays and these are harded encoded in the webserver, therefore, to add new feature or future expansion of this project, new array space should be added to the current global array(s) to accommodate the new future. Changing the current *Idx* for the array will affect the command(s) from the webserver (HMI).

Others:

- Multiple users simultaneously accessing the webserver might leads to system crash or timeout.
- No bumpless transfer is implemented for switching from either close loop control to open loop and vise versal. This is because system is designed to react rapidly to any short circuit faults on the 60Hz grid detected. Implementing bumpless transfer will slow down the response time. Therefore, switching from close loop to open loop and vise versal is used to test this to see how fast the system can respond.
- It is advisable to set the setpoint to zero when powering up the machine to avoid driving the output high.
- If P Control is desired, set KI = 0 and KD = 0
- If PI Control is desired, set KD = 0 and
- If PID Control is desired, set none of the parameters to zero.
- The system status is used for monitoring and debugging, setting wrong values might leads to system crash. The user should be conversant with the function of each fields before setting any value.

- The calibration of the measurement electronic using the Harware offset page should be done with cares due to high voltage. DC voltage is used for the calibration. For the online calibration with the machine, a rectifer can be developed to rectifier the output voltage from the generator to the input of the measurement device.

11 COMMISSIONING AND TESTING

The commissioning and testing for the different control strategies designed and implemented in this project is carried out in this sections along with the webserver designed as the HMI interface for the user.

11.1 OPEN LOOP CONTROL

The setpoint for the open loop control is given in duty cycle (PWM) from the HMI. The duty cycle of *54* gives nominal voltage of *220V* (*phase-to-phase*) for the machine.

When the duty cycle is sets to 60 from the HMI interface, the output voltage seen from the analog and digital multimeters was *225V*. The measurement data obtained was plotted using MATLAB with its wavelet function to filter the noise as shown in *Figure 11-1*.

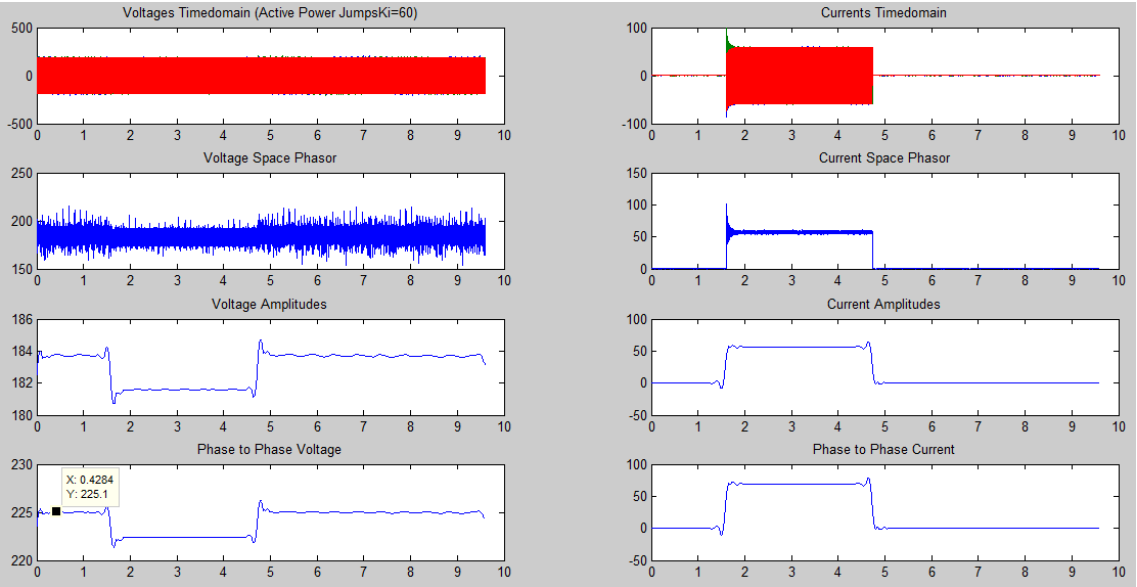


Figure 11-1: Open Loop Control for Duty Cycle of 60.

11.1.1 FUNDAMENTAL, BOUNDARIES AND CHARACTERISTICS

- When operating on open loop the duty cycle is directly written to the output pin connected to the machine through the excitation electonic board.
- Duty cycle range: 0 – 255 (*positive values only*)
- Duty cycle of 54 (PWM) corresponds to nominal voltage of *220V*.
- A negative duty cycle will have no effect on the output of the machine as the code running behind is designed to protect the machine from this.
- The open loop is implemented with no hand-shake with the controllers, this implies there is no bumples transfer between the open loop and the controllers (P, PI and PID).

11.2 PID-CONTROL (P, PI & PID)

11.2.1 ADJUSTMENT OF THE CONTROLLER COEFFICIENTS

Adjusting K_p:

To adjust the $K_p$the following procedures were carried out.

- The K_p range is chosen between 0 and 255

- Set the nominal setpoint measure to 194287 (220V)
- The K_p sets to maximum value of *255* and the division factor ,*a'* in *equation 41* sets to *31* (close to maximum long integer) and decreases the division factor to 23 in a step of 1 until oscillation is observed from the analog and digital voltmeter connected to the output of the machine and from the measurement data ploted using MATLAB as shown in *Figure 11-2*.

  The plots in the figure for current timedomain, current space phasor, voltage amplitude and current amplitude were obtained using *wavelet filter* in MATLAB to filter the noise from the voltage time domain shown.

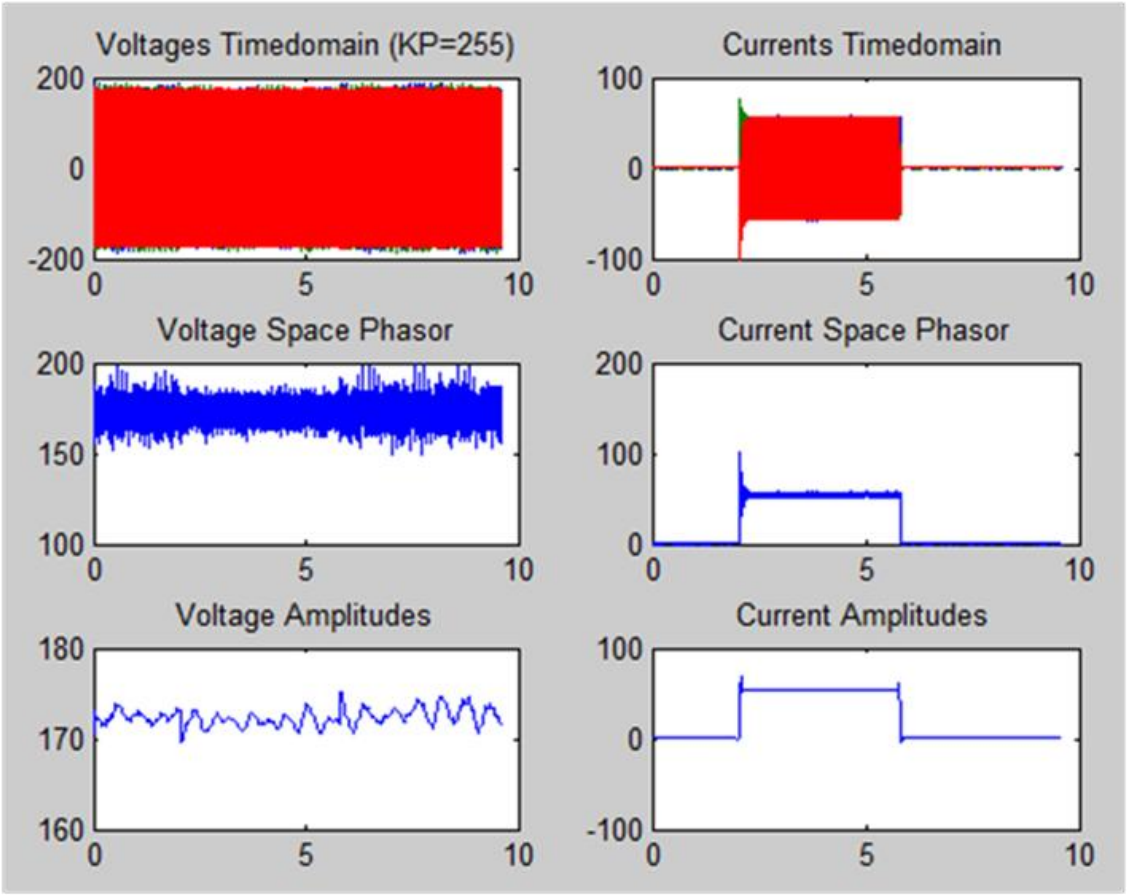


Figure 11-2: Oscillation Observed when the Division Factor 'a' was Set to 23 at 220V Setpoint.

The systematic settings for the K_p and the corresponding findings to determine the next K_p are shown in *Table 11-1*. In each cases a load *4A* (*inductive and resistive loads*) is injected to the grid to observe the output voltage stability based on the K_p setting using the analog voltmeter, digital voltmeter and the measurements data plotted using MATLAB and its inbuilt wavelet function to filter the noise voltage time domain.

Table 11-1: Systematic Steps Determine K_p for the P Controller

Setpoint Measured 194287	Division factor (a)	Kp (min)	Kp (max)	voltage Output for Kp (min)	voltage Output for Kp (max)	Determine next Kp based on findings
220V	23	1	255	Stable at: 199V	Oscillates at: 209V	*(1+255)/2 = 128*
220V	23	1	128	Stable at: 199V	Stable at: 200V	*(128+255)/2 = 192*
220V	23	128	192	Stable at: 200V	Oscillates at: 208V	*(128+192)/2 = 160*
220V	23	128	160	Stable at: 200V	Less oscillation at: 205V	*(128+160)/2 = 144*
220V	23	128	144	Stable at: 200V	Stable at: 202V but with very little dynamic behaviour	*(128+144)/2 = 136*
220V	23	128	136	Stable at: 200V	Stable at 201V	*Kp = 136 chosen*

The systematic procedures for adjusting K_p is shown in *Appendix 15 (P Controller Coeeficient)* using publish function from MATLAB. The *Current Timedomain, Current Space Sphasor, Voltage mplitude and Current Amplitude* shown in the published file are

the output of the wavelet filter from the MATLAB to filter the noise from the *Voltage Timedomain.*

Adjusting K_i:

The systematic procedures for adjusting K_i is shown in *Appendix 15 (PI Controller Coeeficient)* using publish function from MATLAB. The *Current Timedomain, Current Space Sphasor, Voltage mplitude and Current Amplitude* shown in the published file are the output of the wavelet filter from the MATLAB to filter the noise from the *Voltage Timedomain.*

Adjusting K_d:

The systematic procedures for adjusting K_d is shown in *Appendix 15 (PID Controller Coeeficient)* using publish function from MATLAB.

The *Current Timedomain, Current Space Sphasor, Voltage mplitude and Current Amplitude* shown in the published file are the output of the wavelet filter from the MATLAB to filter the noise from the *Voltage Timedomain.*

11.2.2 FINDINGS ON SETPOINT JUMPS

The setpoint jumps are carried out with different controllers in two steps

- *Positive Setpoint Jump: 209V – 231V based on*:
    - Active Power (Resistive Load) - Measurement1 in SD card
    - Positive Reactive Power (Inductive Load) - Measurement3 in SD card
    - Negative Rective Power (Capacitive Load) - Measurement5 in SD card
- *Negative Setpoint Jump: 231V – 209V based on:*
    - Active Power (Resistive Load) - Measurement2 in SD card
    - Positive Reactive Power (Inductive Load) - Measurement4 in SD card
    - Negative Rective Power (Capacitive Load) - Measurement6 in SD card

For each measurements the load was sets to *4A*.

Findings from the experiment carried out on the machine are shown in the following figures. However, only the positive setpoint jumps for the Active Power for P, PI and PID control are shown. Others data for teh respective load are stored in the SD card

attached to the back cover of this report with the MATLAB codes for plottind the data. The MATLAB code can also be found in *Appendix 16* (*Setpoint Jumps*).

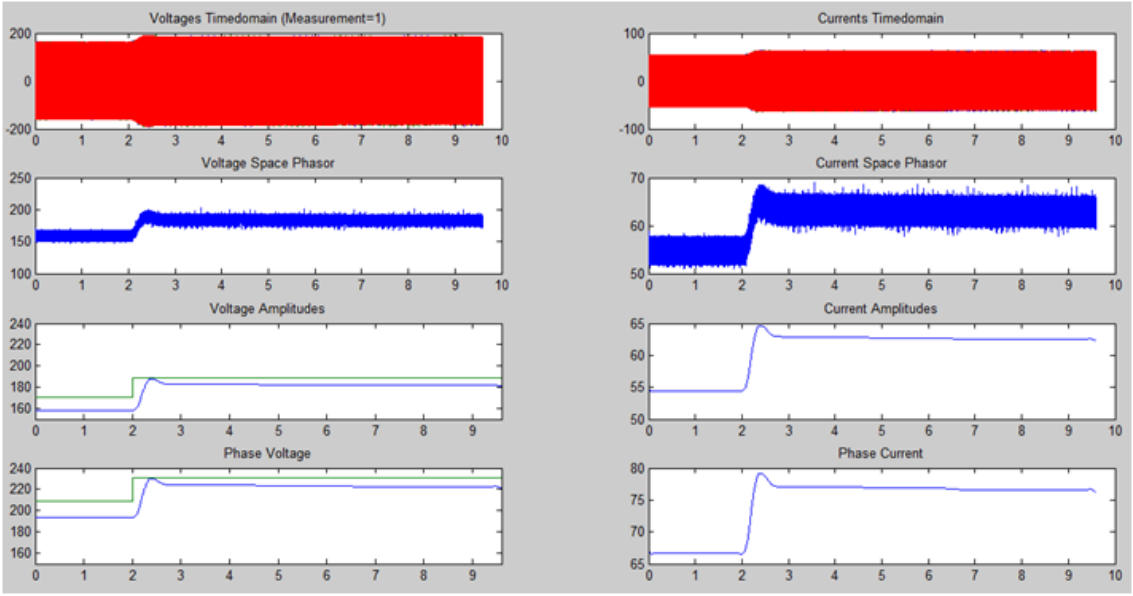


Figure 11-3: Positive Setpoint Jump (209V - 231V) Based on Active Power (Resistive Load) at 4A for P Controller.

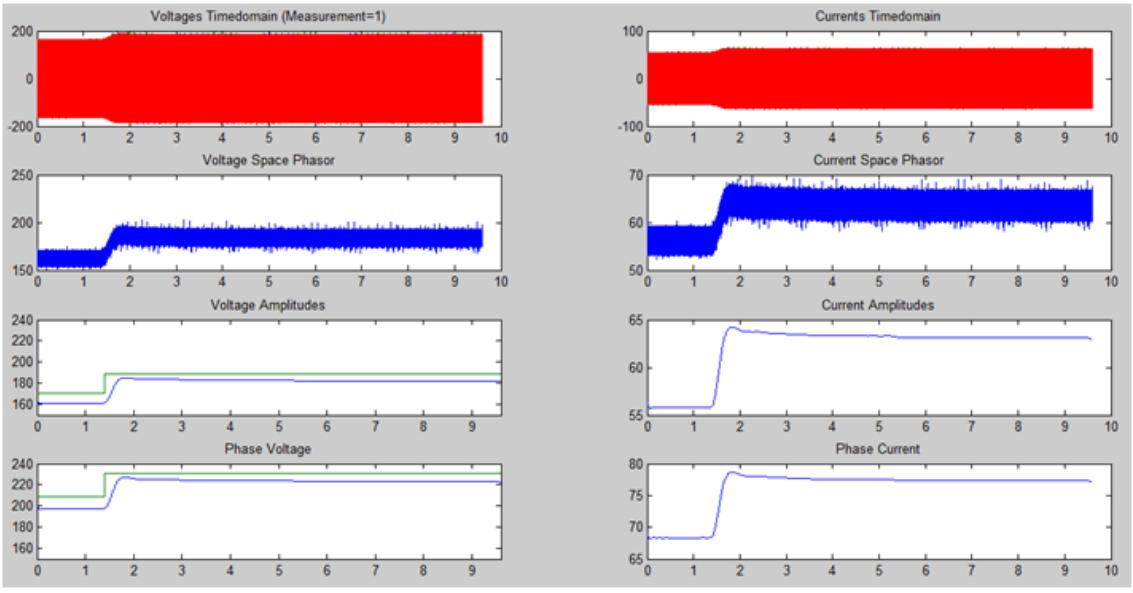


Figure 11-4: Positive Setpoint Jump (209V - 231V) Based on Active Power (Resistive Load) at 4A for PI Controller.

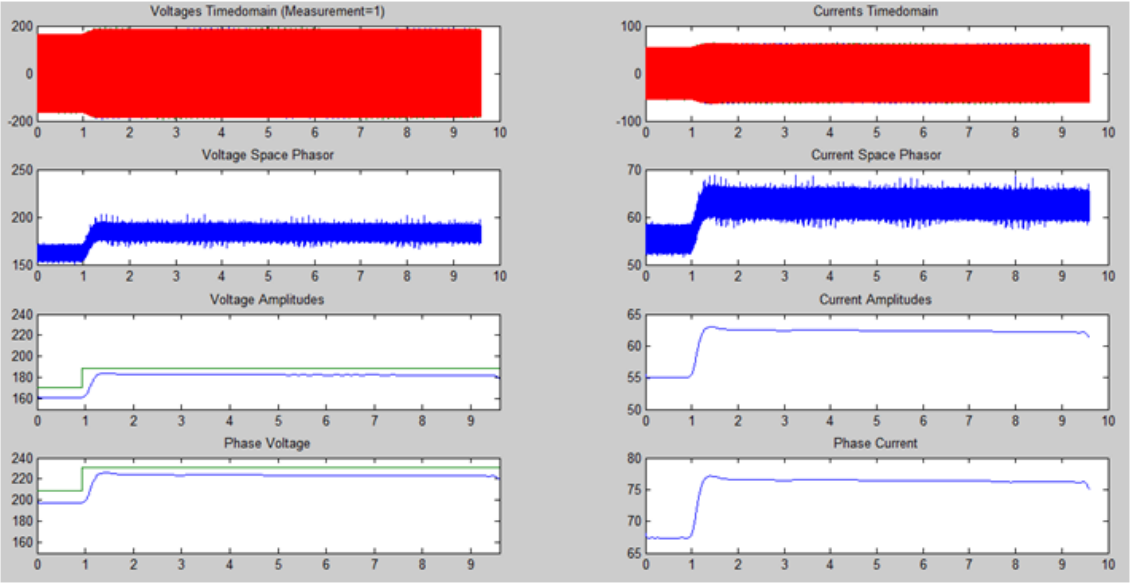


Figure 11-5: Positive Setpoint Jump (209V - 231V) Based on Active Power (Resistive Load) at 4A for PID Controller.

It was seen from *Figure 11-3*, *Figure 11-4* and *Figure 11-5* that the output voltage for the machine does not return exactly to the setpoint. An offset were observed for all the control.

11.2.3 FINDINGS ON ACTIVE POWER JUMPS (RESISTIVE LOAD)

When the phase to phase voltage setpoint was set to *220V*, $K_p = 136$, $K_i = 131$, $K_d = 176$ and the resistive load at 4A connected to the grid was switched on at different time shown in *Figure 11-6*, *Figure 11-7* and *Figure 11-8* the measurement data obtained were plotted using MATLAB with wavelet function to filter the noise as shown in figures.

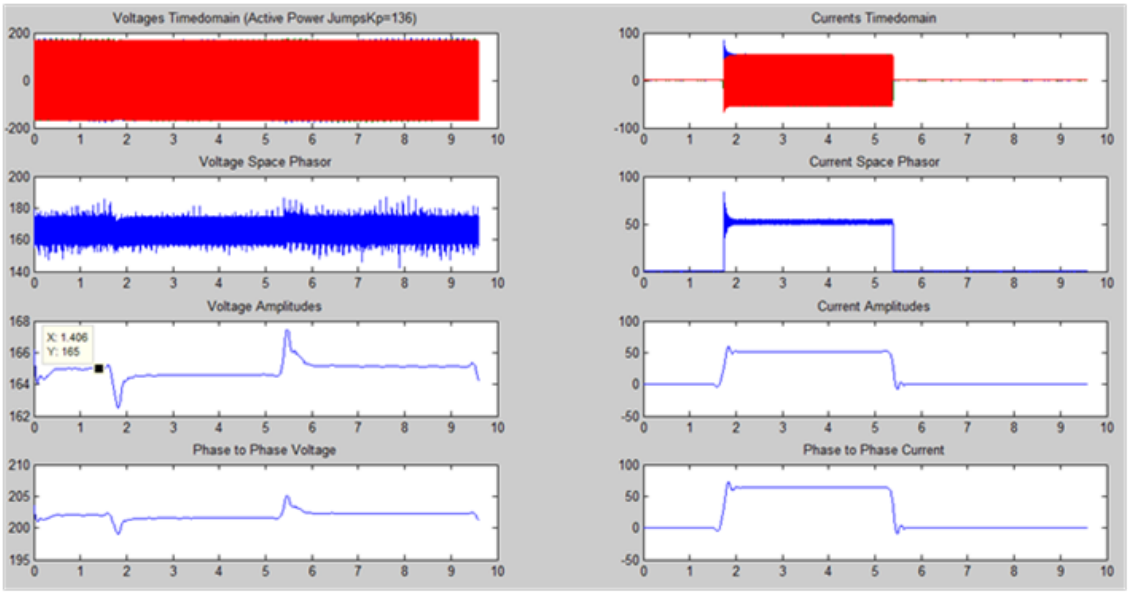


Figure 11-6: The P Controller Behaviour When the Active Power Jumps (Resistive Load) of 4A was Switched on at t = 1.406s (Set: $K_p = 136,\ K_i = 0, K_d = 0$).

As seen from the plot in *Figure 11-6*, when the resistive load (*disturbance*) was switched on at t = *1.406s*, the output voltage dropped from *202V* to *199V (phase-to-phase* and the *P-Controller* return and stabilize the output voltage at *202V* after *500ms* with an offset of *17V* as seen in the figure. When the load was switched off (*disturbance removed*) at *t = 5.0s*, the voltage increases from *202V* to *205V* (an overshoot was observed) and the controller returns and stabilize the voltage at *202V* after *600ms* as seen in *Figure 11-6.*

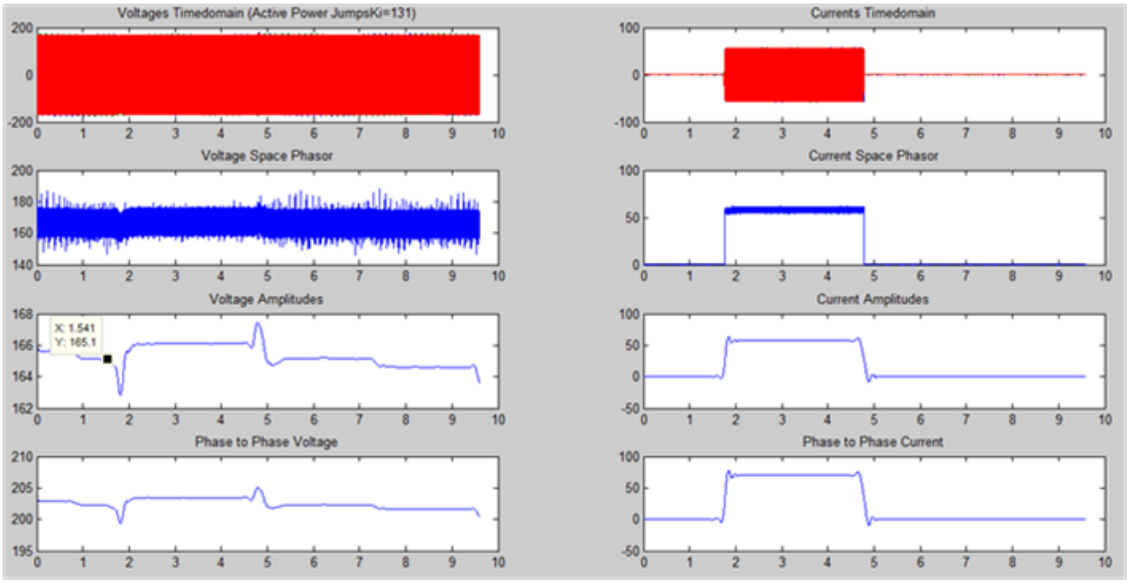


Figure 11-7: The PI Controller Behaviour When the Active Power Jumps (Resistive Load) of 4A was Switched on at t = 1.541s (Set: $K_p = 136,\ K_p = 131,\ K_d = 0$).

As seen from the plot in *Figure 11-7*, when the resistive load (*disturbance*) was switched on at t = *1.541s*, the output voltage decreases from *202V* to *200V* (*phase-to-phase* and the *PI-Controller* return and stabilize the output voltage at *204V*after *500ms* with an offset of *15V* as seen in the figure. When the load was switched off (*disturbance removed*) at *t = 4.498s*, the output voltage increases from *205V* to *206V* (an overshoot was observed) and the controller returns and stabilize the voltage at *204V* after *400ms* as seen in *Figure 11-7*.

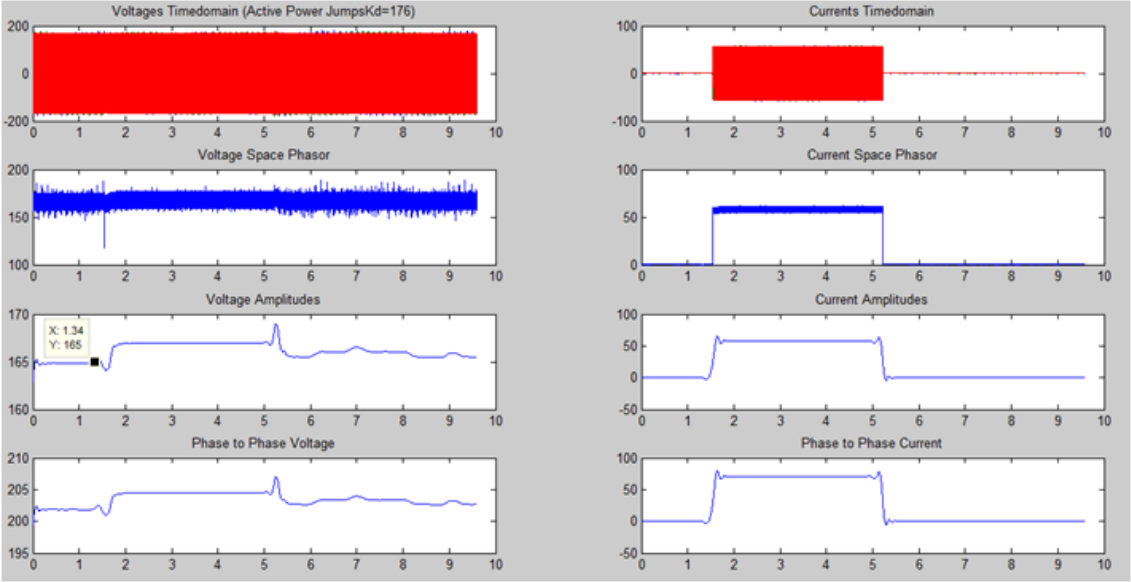


Figure 11-8: The PID Controller Behaviour When the Active Power Jumps (Resistive Load) of 4A was Switched on at t = 1.34s (Set: $K_p = 136,\ K_p = 131,\ K_d = 176$)

As seen from the plot in *Figure 11-8*, when the resistive load (*disturbance*) was switched on at t = *1.34s*, the output voltage dropped from *203V* to *201V (phase-to-phase* and the *PID-Controller* return and stabilize the output voltage at *205V* after *490ms* with an offset of *15V* as seen in the figure. When the load was switched off (*disturbance removed*) at *t =5.0s*, the output voltage increases from *205V* to *207V* (an overshoot was observed) and the controller returns and stabilize the voltage at *204V* after *388ms* as seen in *Figure 11-8.*

Remark:

The MATLAB code for the data analysis for active power jumps, positive reactive power jumps, and negative reactive power jumps are shown in *Appendix 16*. The measurement data and the MATLAB codes is also stored in the SD card (*Filename: Measurement Files*) attached to the back cover of this report. In the MATLAB code replace the ‚*FileName*' in the ‚*Prefix*' with the name of the measurement data (excluding the *fileNo* from the *Prefix*) to be analysed and the ‚*Idx = fileNo*' for example:

Prefix = ' PcontrolActivePowerJumpIdx', the Idx for the file is the *fileNo: 136.*

11.2.4 FINDINGS ON POSITIVE REACTIVE POWER JUMPS (INDUCTIVE LOAD)

When the phase to phase voltage setpoint was set to *220V*, $K_p = 136$, $K_i = 131$, $K_d = 176$ and the inductive load at *4A* connected to the grid was switched on at different time shown in *Figure 11-9*, *Figure 11-10* and *Figure 11-11* the measurement data obtained were plotted using MATLAB with wavelet function to filter the noise as shown in figures.

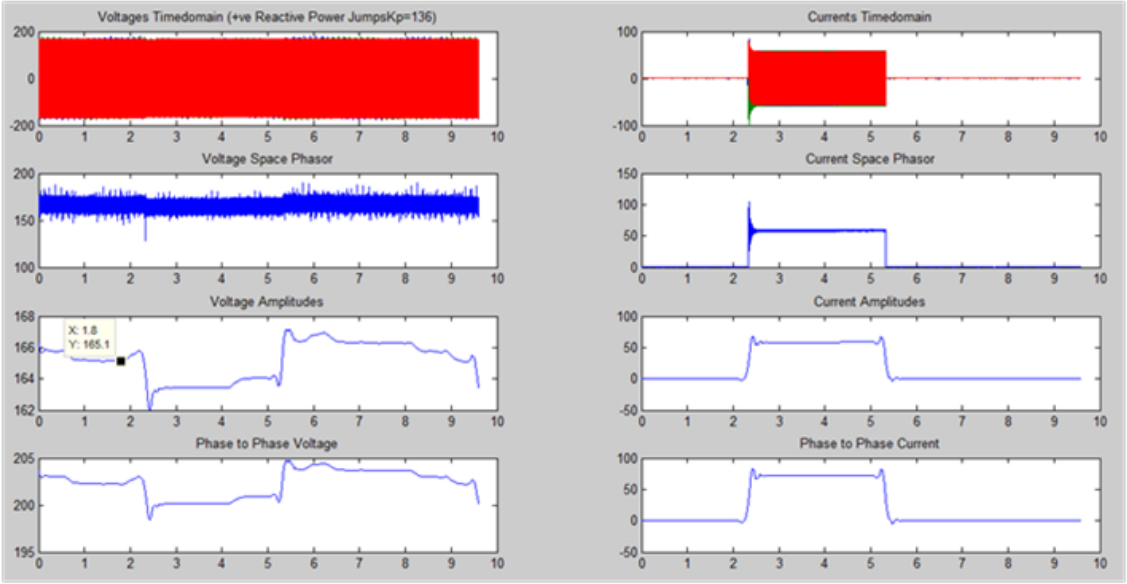


Figure 11-9: The P Controller Behaviour When a Positve Reactive Power Jumps (Inductive Load) of 4A was Switched on at t = 1.8s (Set: $K_p = 136,\ K_i = 0, K_d = 0$).

As seen from the plot in *Figure 11-9*, when the inductive load (*disturbance*) was switched on at t = *1.8s*, the output voltage decreases from *202V* to *199V (phase-to-phase* and the *P-Controller* return and stabilize the output voltage at *202V* after *800ms* with an offset of *17V* as seen in the figure. When the load was switched off (*disturbance removed*) at *t = 5s*, the output voltage increases from *202V* to *204V* (an overshoot was observed) and the controller returns and stabilize the voltage at *202V* after *900ms* with oscillation observed as seen in *Figure 11-9.*

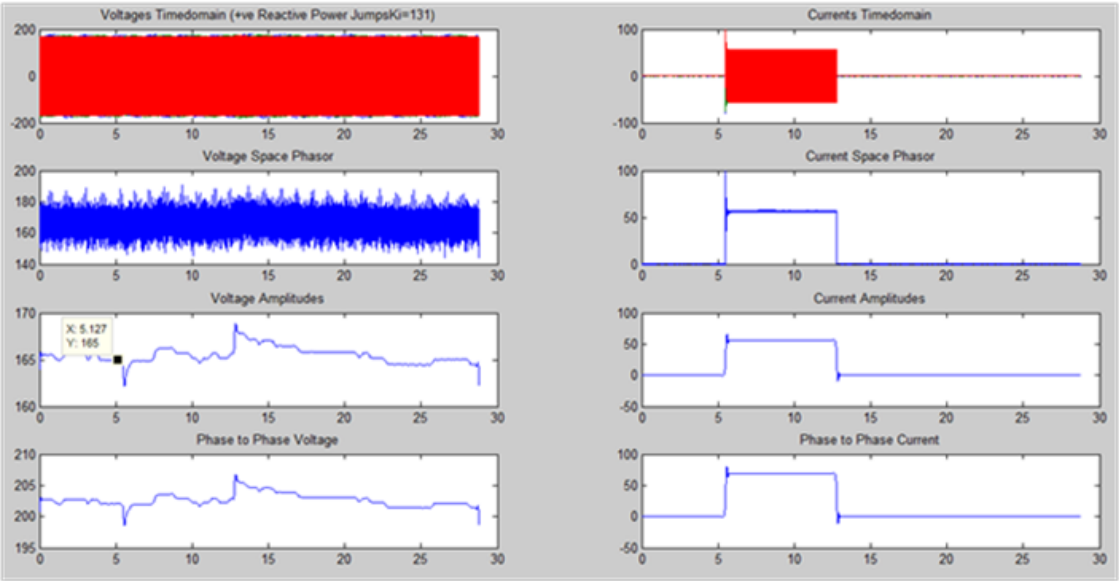


Figure 11-10: The PI Control Behaviour When a Positve Reactive Power Jumps (Inductive Load) of 4A was Switched on at t = 5.127s (Set: $K_p = 136,\ K_i = 131,\ K_d = 0$).

As seen from the plot in *Figure 11-10*, when the inductive load (*disturbance*) was switched on at t = 5.127s, the output voltage dropped from *202V* to *199V (phase-to-phase* and the *PI-Controller* return and stabilize the output voltage at *205V*after *600ms* as seen in the figure. When the load was switched off (*disturbance removed*) at *t = 10.4s*, the output voltage increases from *205V* to *207V* (an overshoot was observed) and the *PI-Controller* returns and stabilize the voltage at *205V* after *400ms* as seen in *Figure 11-10.*

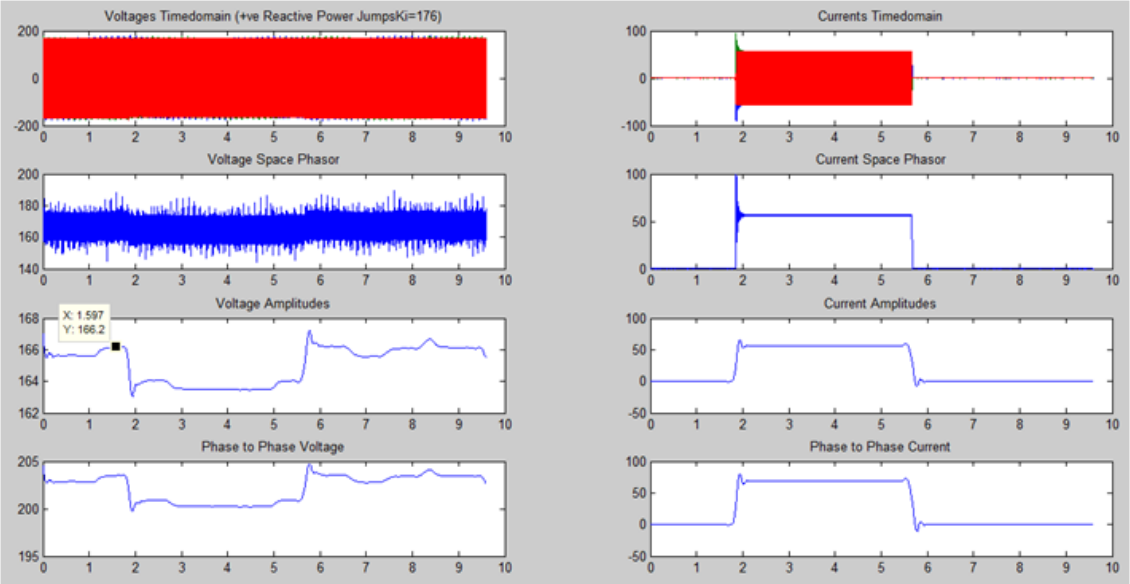


Figure 11-11: The PID Control Behaviour When a Positve Reactive Power Jumps (Inductive Load) of 4A was Switched on at t = 1.597s (Set: $K_p = 136,\ K_i = 131,\ K_d = 176$).

As seen from the plot in *Figure 11-11*, when the inductive load (*disturbance*) was switched on at *t = 1.597s*, the output voltage decreases from *204V* to *200V (phase-to-phase* and the *PID-Controller* returns the output voltage to *201V*after *200ms* as seen in the figure. When the load was switched off (*disturbance removed*) at *t = 5.373s*, the output voltage increases from *201V* to *205V* with an overshoot and the controller returns and stabilizes the voltage at *204V* after *400ms* with oscilation observed as seen in *Figure 11-11*.

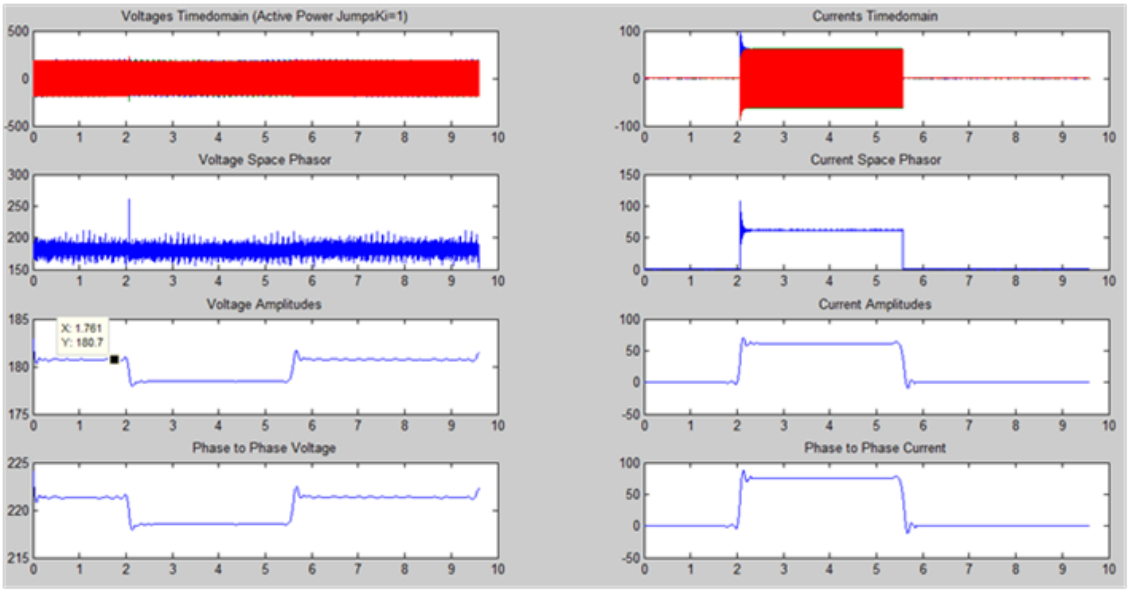


Figure 11-12: The Open Loop Control Behaviour When a Positve Reactive Power Jumps (Inductive Load) of 4A was Switched on at t = 1.761s.

As seen from the plot in *Figure 11-12*, when the inductive load (*disturbance*) was switched on at *t = 1.761s*, the output voltage drops from *222V* to *217V (phase-to-phase* and stabilises at *217V* after *800ms* with an oscillation observed. When the load was switched off (*disturbance removed*) at *t = 5.408s*, the output voltage increases from *217V* to *222V* after *400ms* with an an oscillations observed as seen in *Figure 11-12.*

11.2.5 FINDINGS ON NEGATIVE REACTIVE POWER JUMPS (CAPACITIVE LOAD)

When the phase to phase voltage setpoint was set to *220V*, $K_p = 136$, $K_i = 131$, $K_d = 176$ and the inductive load at *4A* connected to the grid was switched on at different time shown in *Figure 11-13*, *Figure 11-14* and *Figure 11-15* the measurement data obtained were plotted using MATLAB with wavelet function to filter the noise as shown in figures.

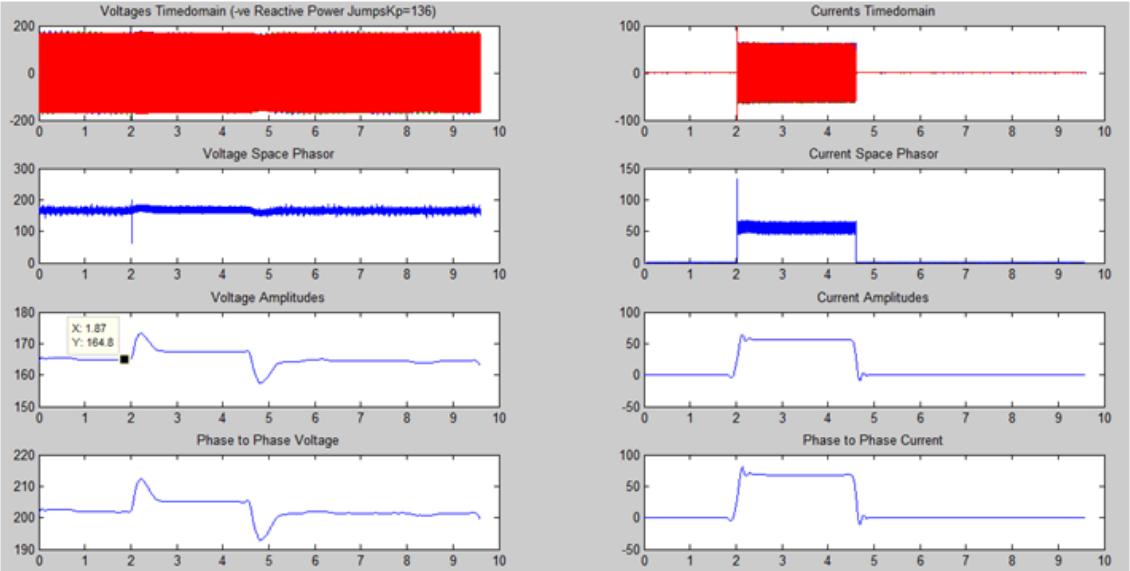


Figure 11-13: The P Controller Behaviour When a Negative Reactive Power Jumps (Capacitive Load) of 4A was Switched on at t = 1.87s (Set: $K_p = 136$, $K_i = 0, K_d = 0$).

As seen from the plot in *Figure 11-13*, when the capacitive load (disturbance) was switched on at t = 1.87s, the output voltage increases from *202V* to *212V (phase-to-phase)* instead of decreasing, and the *P-Controller* return the output voltage to *205V* after *500ms.* When the load was switched off (disturbance removed) at *t = 4.518s*, the output voltage dropped from *205V* to *193V* instead of increasing but the *P-Controller* returns the output voltage to *202V* after *700ms* as seen in the figure.

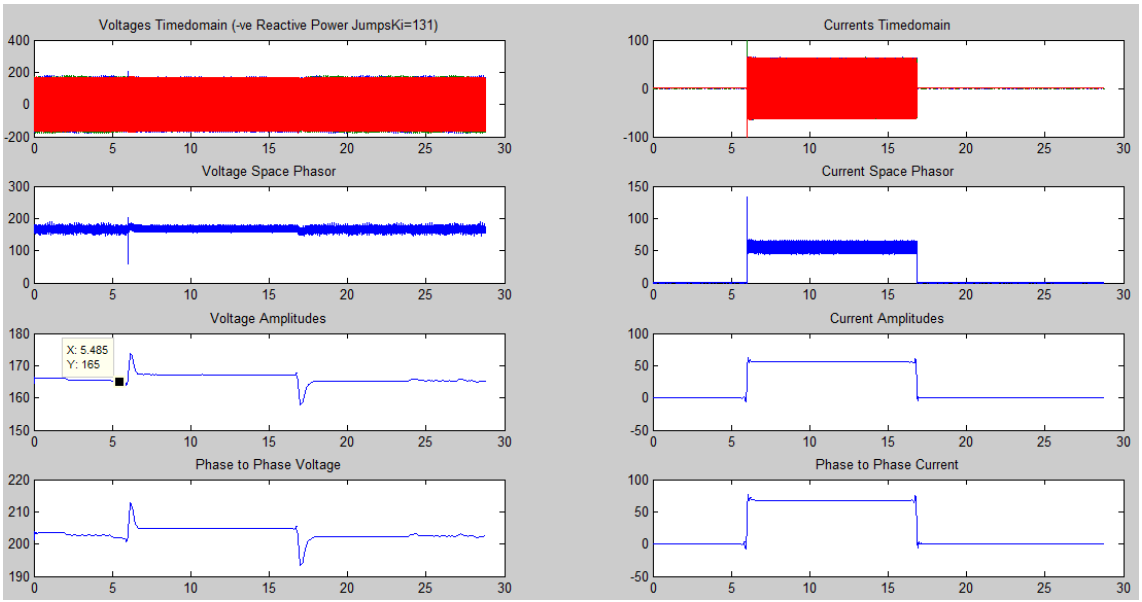


Figure 11-14: The PI Controller Behaviour When a Negative Reactive Power Jumps (Capacitive Load) of 4A was Switched on at t = 5.485s (Set: $K_p = 136$, $K_i = 131, K_d = 0$*).*

As seen from the plot in *Figure 11-14*, when the capacitive load (*disturbance*) was switched on at t = 5. *485*s, the output voltage decreases from *201V* and then increases immediately to *213V (phase-to-phase)* instead of decreasing and the *PI-Controller* return and stabilize the output voltage at *205V* after *700ms.* When the load was switched off (*disturbance removed*) at *t = 16.52s*, the output voltage dropped from *205V* to *194V* instead of increasing but the *PI-Controller* returns and stabilize the output voltage at *205V* after *800ms* as seen in *Figure 11-14.*

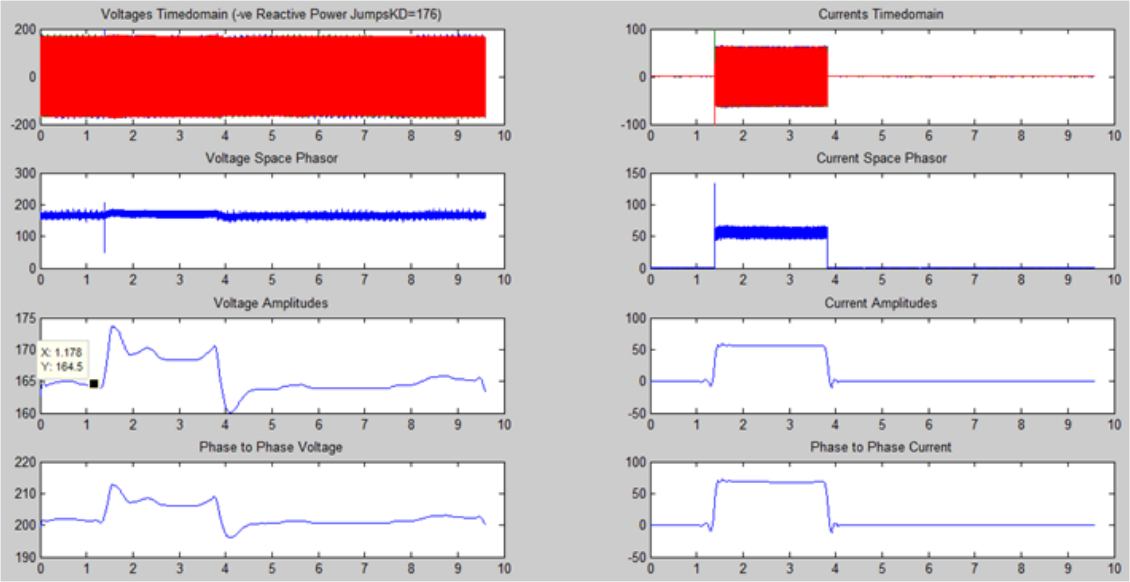


Figure 11-15: The PID Controller Behaviour When a Negative Reactive Power Jumps (Capacitive Load) of 4A was Switched on at t = 1.178s (Set: $K_p = 136$, $K_i = 131, K_d = 0$).

As seen from the plot in *Figure 11-15*, when the capacitive load (*disturbance*) was switched on at *t = 1.178s*, the output voltage increases from *202V* to *213V* (*phase-to-phase*) instead of decreasing and the *PID-Controller* return and stabilises the output voltage to *206V* after *700ms.* When the load was switched off (disturbance removed) at *t = 3.359s*, the output voltage dropped to *196V* instead of increasing and the controller returns the output voltage to *202V* after *1000ms* as seen in the figure.

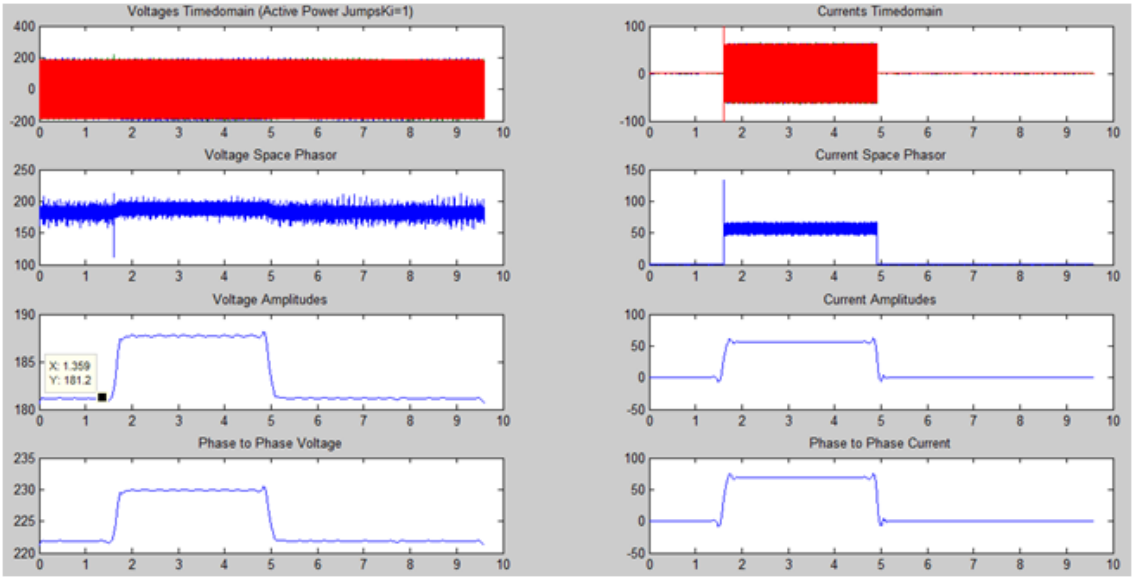


Figure 11-16: The Open Loop Control Behaviour When a Negative Reactive Power Jumps (Capacitive Load) of 4A was Switched on at t = 1.359s.

As seen from the plot in *Figure 11-16*, when the capacitive load (*disturbance*) was switched on at *t = 1.359s*, the output voltage increases from *222V* to *230V* (*phase-to-phase*) instead of decreasing and stabilizes at *230V after 300ms.* When the load was switched off (*disturbance removed*) at *t = 4.711s*, the output voltage dropped to *222V* after *300ms* instead of increasing as seen in the figure.

Table 11-2 shows a brief summary of the findings discussed in this section.

Table 11-2: Brief Summary of the Findings.

	P Control	PI Control	PID Control	Open Loop Control
Positive Setpoint Jumps with Active Power (Resistive Load) (*209V– 231V*)	Voltage stabilizes at *224V*	Voltage stabilizes at *225V*	Voltage stabilizes at *224V*	Voltage stabilizes at *222V*
Negative Setpoint Jumps with Active Power (Resistive	Voltage stabilizes at *197V*	Voltage stabilizes at *198V*	Voltage stabilizes at *198V*	Voltage stabilizes at *196V*

Load)(*231V– 209V*)				
Positive Setpoint Jumps with positive Reactive Power (Inductive Load) (*209V–231V*)	Voltage stabilizes at *224V*	Voltage stabilizes at *223V*	Voltage stabilizes at *225V*	Voltage stabilizes at *222V*
Positive Setpoint Jumps with Negative Reactive Power (Capacitive Load)(*209V– 231V*)	Voltage stabilizes at *227V*	Voltage stabilizes at *226V*	Voltage stabilizes at *225V*	Voltage stabilizes at *229V*
Negative Setpoint Jumps with positive Reactive Power (Inductive Load) (*231V– 209V*)	Voltage stabilizes at *198V*	Voltage stabilizes at *196V*	Voltage stabilizes at *194V*	Voltage stabilizes at *197V*
Negative Setpoint Jumps with Negative Reactive Power (Capacitive Load)(*231V– 209V*)	Voltage stabilizes at *196V*	Voltage stabilizes at *199V*	Voltage stabilizes at *195V*	Voltage stabilizes at *195V*
Response Time when Active load switched on (Resistive load)	*500ms*	*500ms*	*490ms*	*Data not available*
Response Time when Active load switched off (Resistive load)	*600ms*	*400ms*	*388ms*	*Data not available*
Response Time when Positive Reactive load switched on	*800ms*	*600ms*	*200ms*	*800ms*

(Inductive load)				
Response Time when Positive Reactive load switched off (Inductive load)	*900ms*	*400ms*	*400ms*	*400ms*
Response Time when Negative Reactive load switched on (Capacitive load)	*500ms*	*700ms*	*700ms*	*300ms*
Response Time when Negative Reactive load switched off (Capacitive load)	*700ms*	*800ms*	*1000ms*	*300ms*
Impact of Active Load (Resistive load)	*When the load is switched on:* Output voltage drops to*199V,* the controller returns and stabilized output voltage to *202V.* *When the load is switched off:* Output voltage increases to *205V,* the	*When the load is switched on:* Output voltage drops *200V,* the controller returns and stabilized output voltage to *204V.* *When the load is switched off:* Output voltage increases to *206V,* the controller returns and stabilized the voltage at *204V.*	*When the load is switched on:* Output voltage drops *201V,* the controller returns and stabilized output voltage to *205V.* *When the load is switched off:* Output voltage increases to *207V,* the controller returns and stabilized the voltage at *205V.*	*Data not available*

	controller returns and stabilized the voltage at *202V*.			
Impact of Positive Reactive load (Inductive load)	*When the load is switched on:* Output voltage drops to *199V*, the controller returns and stabilized output voltage to *202V*. *When the load is switched off:* Output voltage increases to *204V*, the controller returns and stabilized the voltage at *202V*.	*When the load is switched on:* Output voltage drops, the controller returns and stabilized output voltage to *205V*. *When the load is switched off:* Output voltage increases, the controller returns and stabilized the voltage at *205V*.	*When the load is switched on:* Output voltage drops, controller could not return the output to the *204V* (output stays below setpoint). *When the load is switched off:* The controller returns and stabilizes the voltage at 204V.	*When the load is switched on:* Output voltage drops from *222V* to *217V* and remains without returning to setpoint *222V*. *When the load is switched off:* Output voltage increases to *222V* and remains.
Impact of Negative Reactive load (Capacitive load)	*When the load is switched on:* Output voltage increases, the controller returns and	*When the load is switched on:* Output voltage increases, the controller returns and stabilizes the	*When the load is switched on:* Output voltage increases, the controller returns and stabilizes the voltage to	*When the load is switched on:* Output voltage increases from *222V* to *230V* and

	stabilizes the voltage to *202V* *When the load is switched off*: Output voltage decreases, the controller returns and stabilizes the voltage to *202V*	voltage to *205V* *When the load is switched off*: Output voltage decreases, the controller returns and stabilizes the voltage to *205V*	204V *When the load is switched off*: Output voltage decreases, the controller returns and stabilizes the voltage to *202V*	remains there. *When the load is switched off*: Output voltage drops to *222V* and remains there.

11.2.6 CONNECTION WITH WIND TURBINE

When the wind turbine was connected to the 60Hz grid as shown in *Figure 11-17*

with a variable load connected to the grid between the turbine and the machine, the load was made to increases or decreases. The voltage at *point A* of the 60Hz machine was to be kept constant for any load conditions. In this experiment P Control was used.

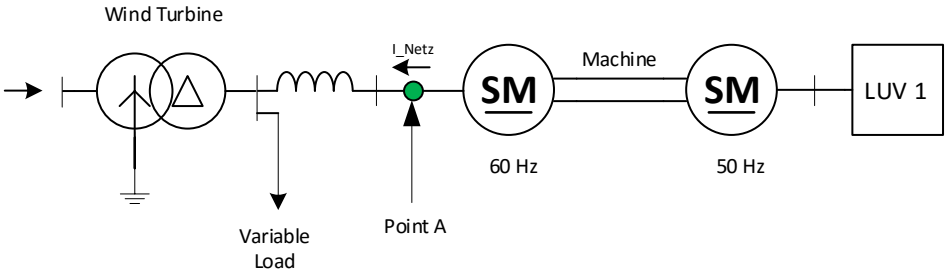


Figure 11-17: Connection of the Wind Turbine to the 60Hz Grid with Variable Load.

The measurement data obtained was analyzed and plotted using MATLAB and its wavelet function as shown in *Figure 11-18* to *Figure 11-22*.

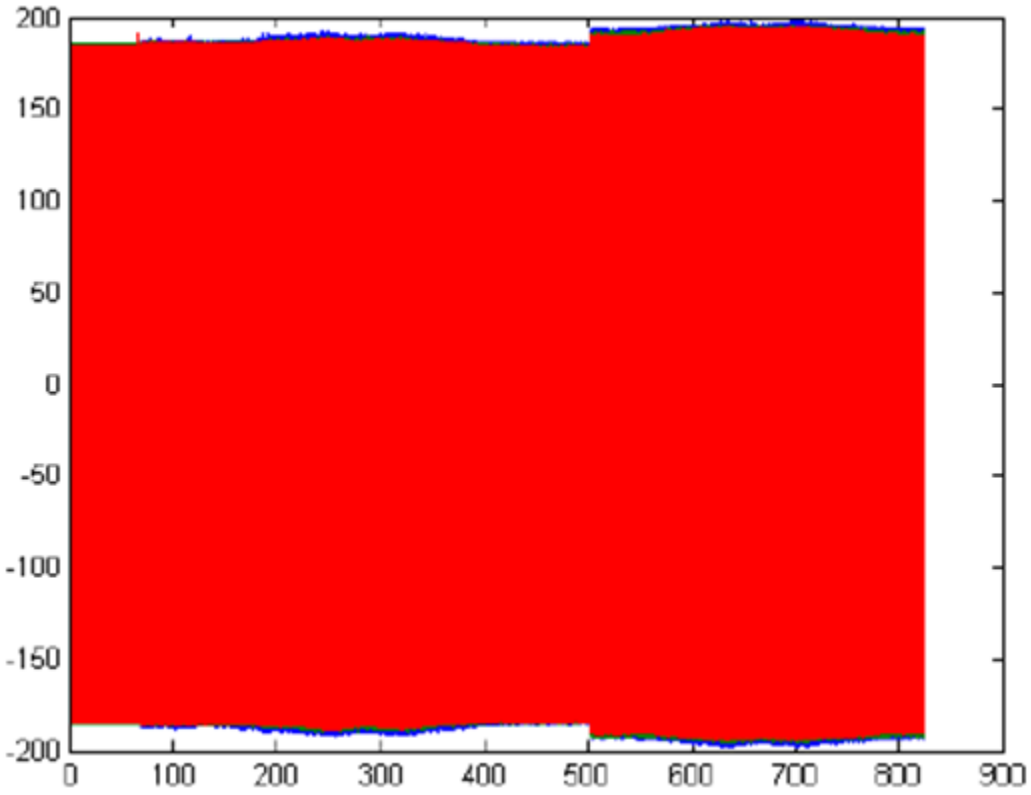


Figure 11-18: Phase Voltage (Phase-Neutral) Measured at Point A of the Machine when the Wind Turbine is Conneted to the 60Hz Grid

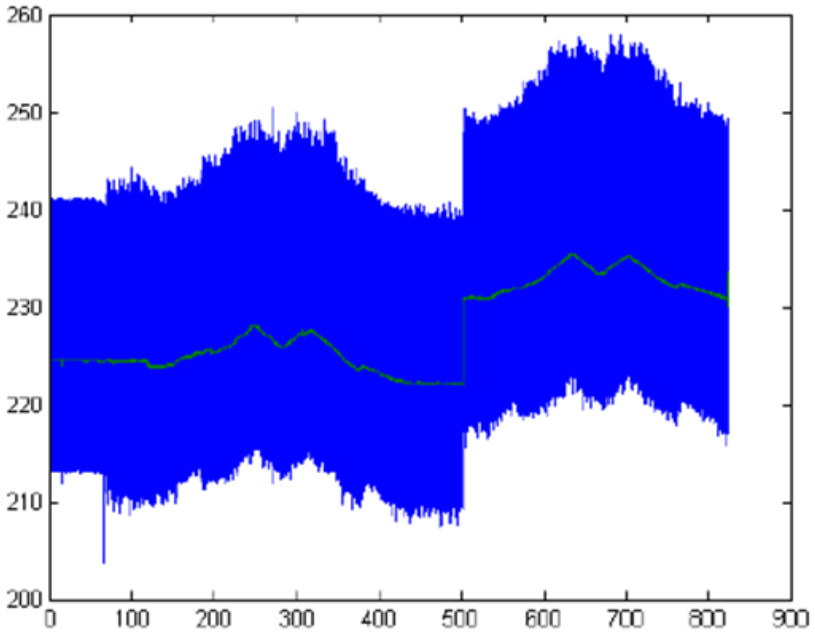


Figure 11-19: Voltage Space Phasor and its Filtered Value Measured at Point A of the Machine when the Wind Turbine is Conneted to the 60Hz Grid.

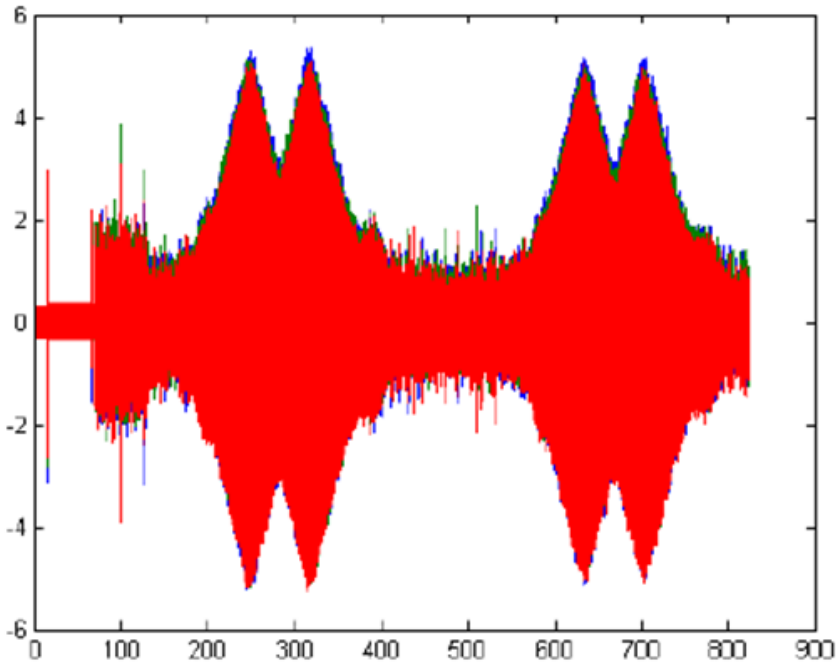


Figure 11-20: Phase Current for RST Measured at Point A of the Machine when the Wind Turbine is Conneted to the 60Hz Grid.

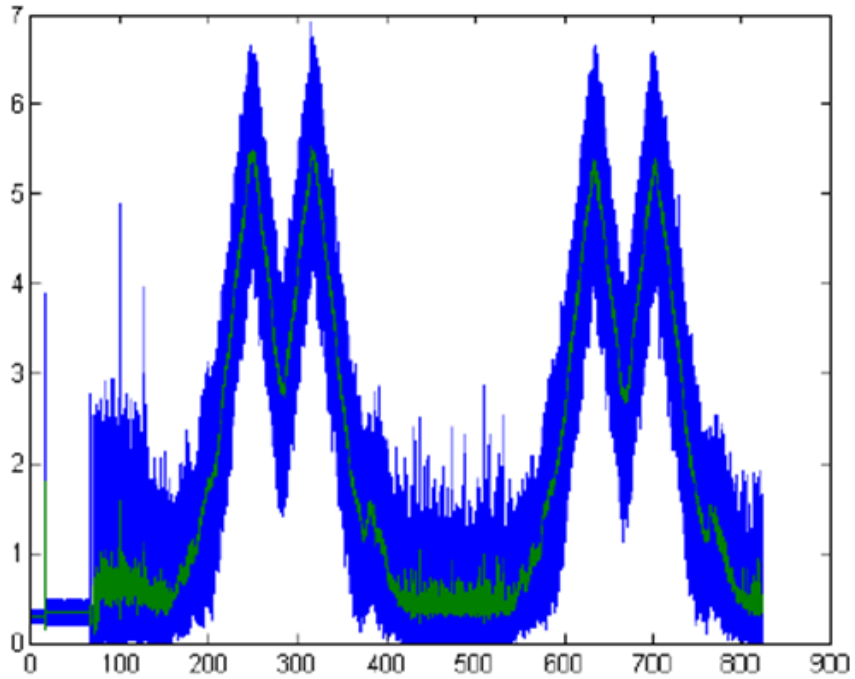


Figure 11-21: Current Space Phasor for RST and it Filtered Value Measured at Point A of the Machine when the Wind Turbine is Conneted to the 60Hz Grid.

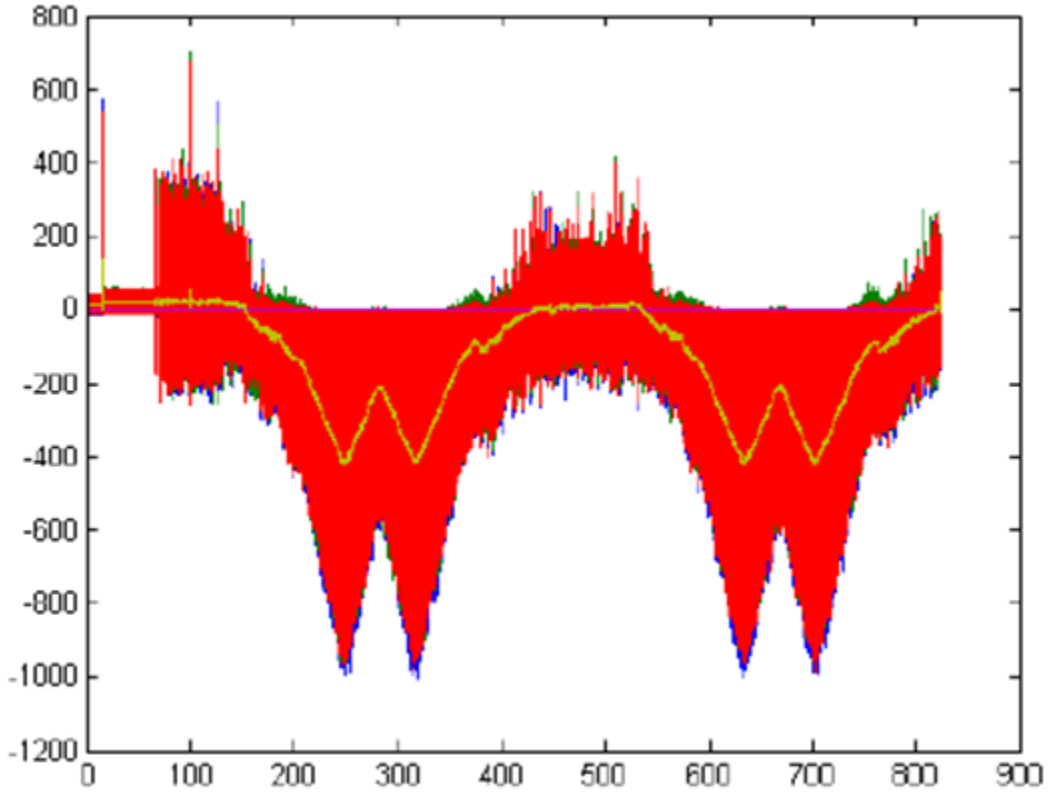


Figure 11-22: Power Flow and its Filtered Value Measured at Point A of the Machine when the Wind Turbine is Conneted to the 60Hz Grid.

As seen from *Figure 11-22* between *0 – 150s* there is positive power flow at *point A* which indicates that the power supplied by the wind turbine is not enough to feed the variable load and this results in outflow of power from the machine to the load and the wind turbine, however, as seen in *Figure 11-19* the *P Control* stabilizes the voltage at within this period for the increased load.

Between *150 – 420s* in *Figure 11-22* there was a negative power flow at *point A* which implied an infeed of power by the 60Hz machine however, this leads to an increase in voltage as seen in *Figure 11-19* but the *P Control* tries to stabilize the voltage at to the setpoint with an offset of *2V* from the setpoint.

At time *502s* shown in *Figure 11-19* the machine was switched to an open loop control under the same load conditions and it was seen from *Figure 11-19* that the change in voltage from the setpoint in an open loop control was *3V*. The P Control was able to reduce the effect of the load by *1V*. The MATLAB code is shown in *Appendix 17*.

12 INTERPRETATION OF FINDINGS

From the figures and *Table 11-2* (Summary of findings) it was seen that the PI control increases and stabilizes the output voltage of the machine at a higher voltage close to the setpoint with the different load conditions compare to P and PID controls. The non linear behaviour of the machine contributes to the inability of the controllers to return the voltage to the exact setpoint. Furthermore, the machine is designed to withstand high or large short circuit faults on the grid and this is observed from the open loop control as there is no significant different from the closed loop control.

It was also seen from *Table 11-2* that it takes aproximately *500 – 700ms* for the P control to stabilize the output voltage of the machine for the capacitive load (*Negative Reactive load*) and *700 – 800ms* for PI control and *700 - 1000ms* for PID control. This implies that when fast response is desired for a capacitive load, P control would be the right choice based on the findings.

For the inductive load (*Reactive load*) as shown in *Table 11-2* it takes approximately *800 – 900ms* for the P control to stabilize the output voltage of the machine and *400 – 600ms* for PI control and *200 – 400ms* for PID control. This indicates that for the inductive load where fast response is required, PID control would be the right choice based on the findings.

Table 11-2 also shows that for a resistive load (Active power) it takes approximately *500 – 600ms* for the P control to stabilize the output voltage of the machine and *400 – 500ms* for the PI control and *388 – 490ms* for the PID control. This shows that for resistive load where fast response is required, PID control would be the right choice based on the findings. However, the PI controller stabilizes the output voltage of the machine close to the setpoint with an average response of *600ms* seen from *Table 11-2*.

When the wind turbine was connected to the 60Hz grid with the machine and the variable load it was seen from *Figure 11-19* that the *P Control* was able to stabilize the voltage with an offset of *2V* compare to open loop control with an offset of *3V*. This implies that the *P Control* works.

13 CONCLUSION

This work covers the introduction for the small scale 60 Hz enegery system , the project description and its perculiarity, a brief literature review, different measurements approaches and findings, the computation approaches, basic theoritical background, the excitation power electronic with RS232 to serial communication, different control strategies and logic, coding, webserver (HMI), testing and commissioning and findings. The report also contains the user guide, technical documentation and the various software used for proper handling shown in *Appendix 18*.

The fuzzy logic control could not be implemented due to time constraint, however, the platform has been design in the *DoControl* function and in the main for this to be implemented in future.

From the findings, the controllers were able to stabilize the voltage close to the setpoint with different load conditions and the connection with the wind turbine.

The selection of any of the control strategies is purely based on the use case and the customer's interest as different control strategies have their perculiar feature, advantages and disadvantages depending on the application and the use case.

14 REFERENCES

Arduino (2016, 07th September, 2015). "Arduino Software (IDE)." Retrieved 26th January, 2016, from https://www.arduino.cc/en/Guide/Environment.

Arduino (2016). "Arduino Uno and Genuino Uno Board." Retrieved 8th May, 2016, from https://www.arduino.cc/en/main/arduinoBoardUno.

Arduino (2016). "PWM." Retrieved 11th May, 2016, from https://www.arduino.cc/en/Tutorial/PWM.

Atmel (2015). "ATMEL 8-Bit Microcontroller with 4/8/16/32KBytes In-System Programmable Flash Datasheet." Retrieved 8th May, 2016, from http://www.atmel.com/images/atmel-8271-8-bit-avr-microcontroller-atmega48a-48pa-88a-88pa-168a-168pa-328-328p_datasheet_complete.pdf.

Atmel, C. (2009). ATmel Datasheet 8-bit AVR Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash.

CONRAD (2016). "Arduino Board Ethernet Platine 65145 ATMega328 ". Retrieved 9th May, 2016, from https://www.conrad.de/de/arduino-board-ethernet-platine-65145-atmega328-191803.html.

Futureelectronics (2016). "IGBT." Retrieved 6th May, 2016, 2016, from http://www.futureelectronics.com/en/transistors/igbt-transistor.aspx.

Helge, L. (2016). 50/60Hz Excitation Board Design for Synchronous Machine.

Kundur, P. (1994). Power system stability and control, Power System Engineering Series. USA, McGraw-Hill, inc.

Manohar, S. S. D. G. (2012). "Literature Review on Voltage stability phenomenon and Importance of FACTS Controllers In Power system Environment."

Mousavi, O. A. (2011). Literature Survey on Fundamental Issues of Voltage and Reactive Power Control

Sweeney, G. (2012). "Interrupt-Driven Analog Conversion With an ATMega328p." Retrieved 9th May, 2016, from http://www.glennsweeney.com/tutorials/interrupt-driven-analog-conversion-with-an-atmega328p.

Wikipedia (2016, 6th January, 2016). "Duty cycle." Retrieved 30th January, 2016, 2016, from https://en.wikipedia.org/wiki/Duty_cycle.

Wikipedia (2016, 4th May, 2016). "Insulated-gate bipolar transistor." Retrieved 5th May, 2016, from https://en.wikipedia.org/wiki/Insulated-gate_bipolar_transistor.

Wikipedia (2016, 8th May, 2016). "Interrupt." Retrieved 8th May, 2016, from https://en.wikipedia.org/wiki/Interrupt.

Wikispaces (2016). "Arduino-PWM-Frequency." Retrieved 7th February, 2016, 2016, from https://arduino-info.wikispaces.com/Arduino-PWM-Frequency.

15 APPENDICES

Contents

15.1 APPENDIX 1: THESIS TOPIC DESCRIPTION

Telemark University College

Faculty of Technology

FMH606 Master's Thesis

Title: Monitoring 50/60Hz grid coupling – A study in conjunction with wind-energy feed to main grids

Jade University of Applied Sciences: Helge Lorentzen & Prof. Josef Timmerberg

TUC supervisor: Saba Mylvaganam

External partner: Jade university of Applied Sciences (Mr. Helge Lorenzen & Prof. Josef Timmerberg)

Task description: A small scale energy systems is operated for teaching and research purpose. This system is to be upgraded to improve possibilities for experiments especially with 60Hz grids. To get the 60Hz frequency stable grid, two mechanically coupled synchronous machines are used. The task is to stabilize the voltage of the 60Hz grid connection.

The following tasks are assigned to the candidate:

1. Brief literature review covering basics of system behaviour of 3-phase electrical power grids and voltage control using synchronous machines.

2. Identify the measuarands involved in the field as well as the corresponding ones in the test lab. Discuss the limitations in the scaling up process from the test lab to the field.
3. Investigate different characteristic values for variables of three phase power systems (e.g. rms values, space phasors, fundamental amplitude estimation) by own implementation to compute.
4. Use sensor data fusion (e.g. voltage and current sensors) to get higher level information like active and reactive power as an input for control strategies.
5. Implement different strategies (e.g. Fuzzy logic control and P, PI, PID controls) to stabilize the voltage for the 60Hz point of common coupling using the sensor data.
6. Use given µ-controller system (Arduino) and MATLAB for investigations.
7. Create a systematic comparison of the findings for different data acquisition and control strategies.
8. Interpret/discuss the results/findings
9. Delivery of written thesis following guidelines from TUC

Task background:

The experimental platform is located in the european 50Hz interconnected system and also used for testing of wind turbines in 60Hz grids. The equipment set enables in frequency a very rigid coupling and thus a high frequency stability of the 60Hz grid. It significantly reduces the available short-circuit power and thus places increased demands and voltage control to get a really rigid busbar (slack).

Student category:

This project is specifically defined for SCE student Achema Egbunu

Practical arrangements:

This work focuses on measurements, data fusion related to wind energy generation. The work will involve signal processing, usage of microcontrollers and high-level language programming.

Helge Lorenzen is the project team leader and will be involved in some of the planning and supervision in the project. Practical implementation and investigations will be performed in his laboratory at Jade University of Applied Sciences, Germany.

Filename: `Saba_Mylvaganam_master_project.rtf`

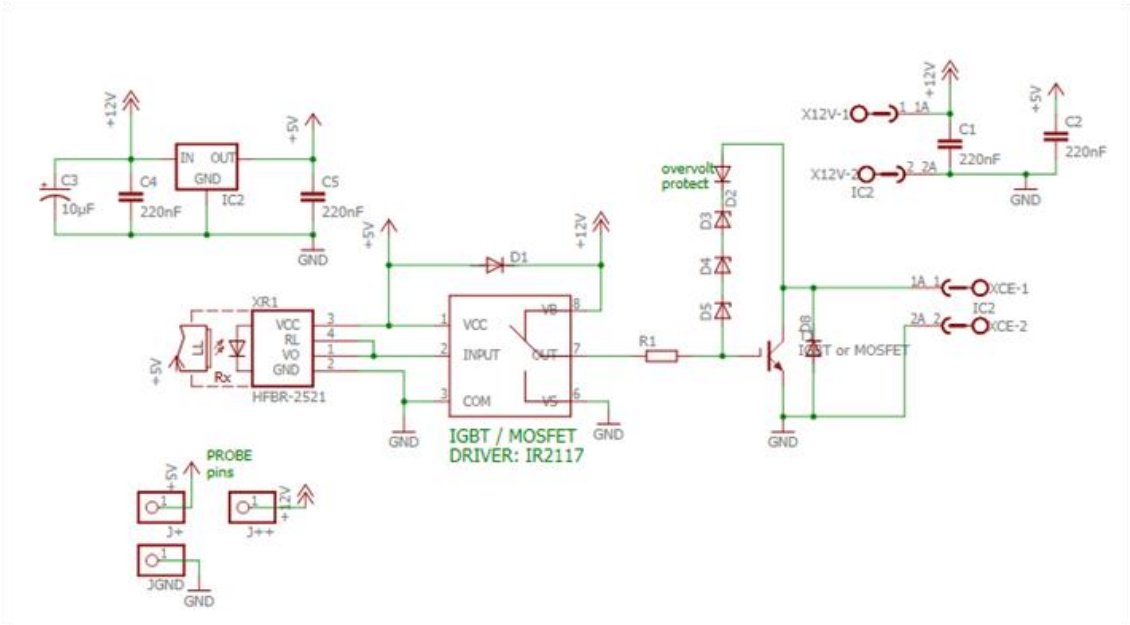
Signatures:

Student (date and signature):

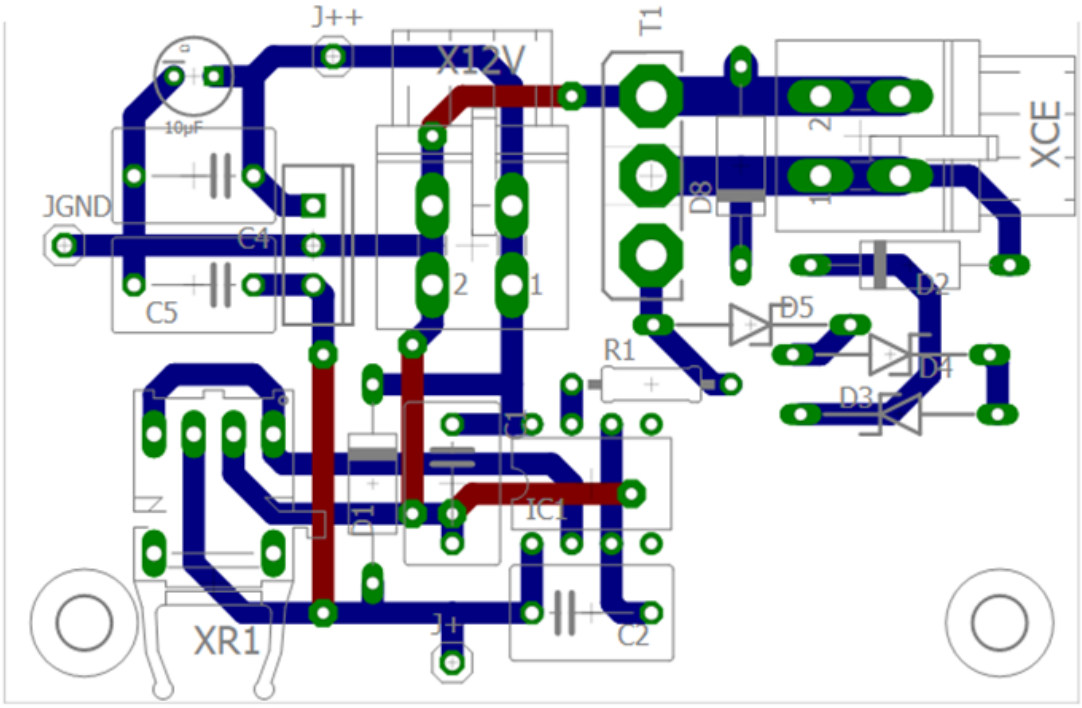
Supervisor (date and signature):

15.2 APPENDIX 2: EXCITATION ELECTRONIC CIRCUIT

The Excitation Board Circuit Diagram, PCB Design and its Components designed by *Helge Lorenzen* using *EAGLE Version 7.5.0* for the project. The components and their values along with the circuit are also stored in the SD Card attached to the back cover of this report.



The Excitation Board Designed By Helge Lorenzen.



PCB for the Excitation Board Designed By Helge Lorenzen.

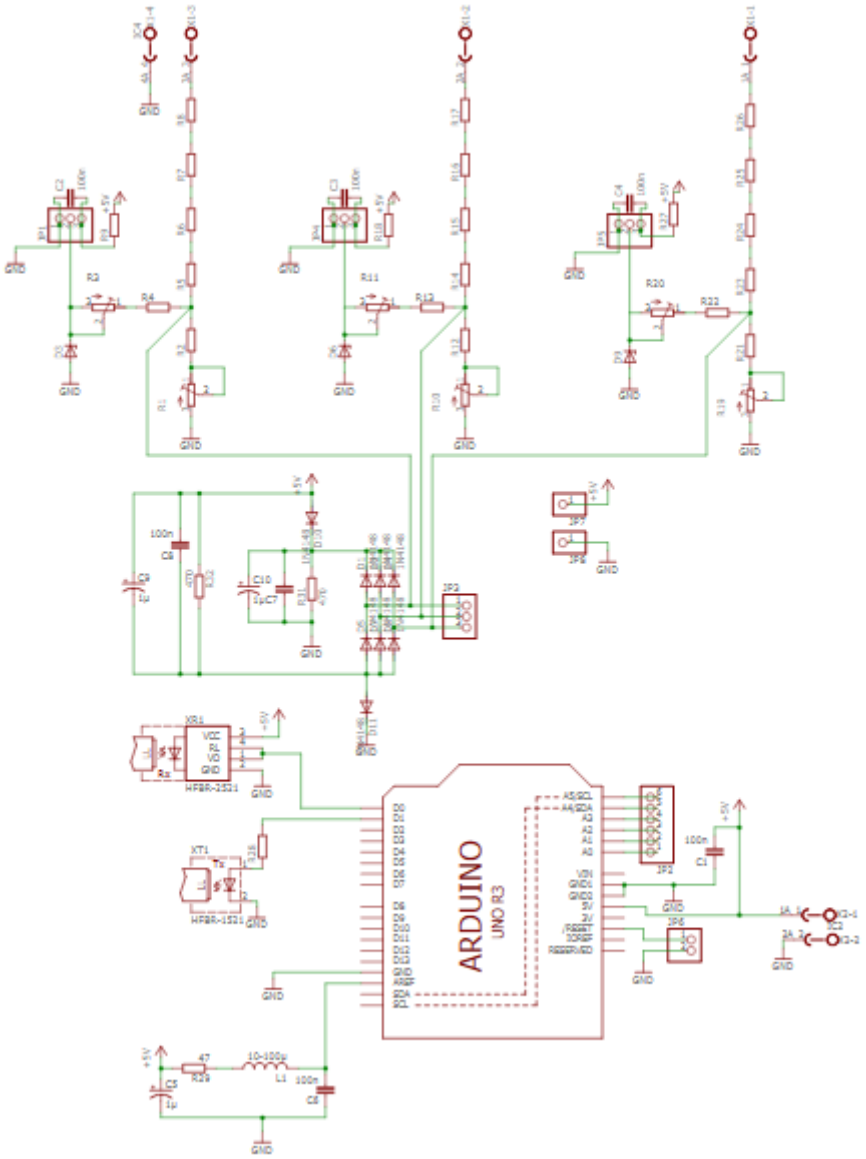
Components and its values.

Part	Value	Package	Library	Position (inch)	Orientation
C1	220nF	C075-063X106	Rcl	(1.125 0.55)	R270
C2	220nF	C075-063X106	rcl	(1.475 0.225)	R0
C3	10µF	CPOL-RADIAL-10UF-25V	SparkFun	(0.475 1.575)	R0
C4	220nF	C075-063X106	Rcl	(0.475 1.325)	R180
C5	220Nf	C075-063X106	Rcl	(0.475 1.05)	R180
D1		D-12.5	Diode	(0.925 0.55)	R270
D2		D-12.5	Diode	(2.275 1.1)	R0
D3		ZDIO12.5	Diode	(2.25 0.725)	R0
D4		ZDIO12.5	Diode	(2.225 0.875)	R180

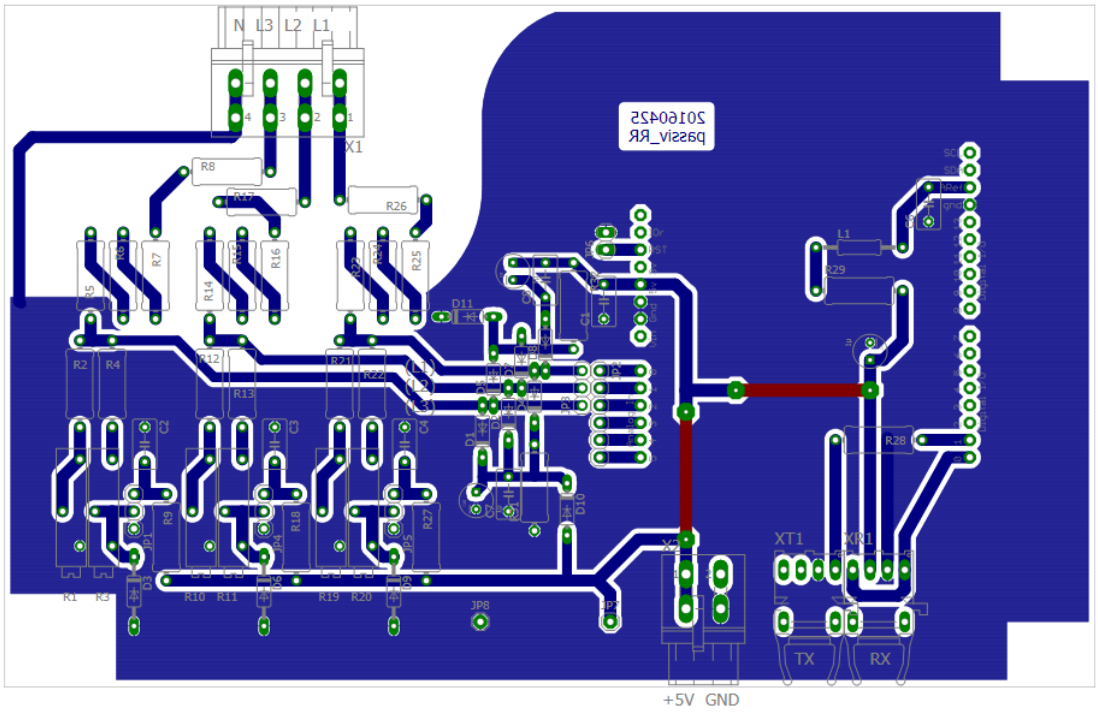
D5		ZDIO12.5	Diode	(1.875 0.95)	R180
D8		D-12.5	Diode	(1.85 1.35)	R90
IC1	IR2117	DIL08	Ir	(1.475 0.55)	R0
IC2		TO220-IGO	SparkFun	(0.775 1.15)	R270
J+		1X01	Pinhead	(1.125 0.1)	R0
J++		1X01	Pinhead	(0.825 1.625)	R0
JGND		1X01	Pinhead	(0.15 1.15)	R0
R1		0207/10	Resistor	(1.625 0.8)	R0
T1	IGBT or MOSFET	TO-247-3V	HL_IGBT_ etc	(1.625 1.325)	R90
X12V	IC2	IC2	con-phoenix-508	(1.175 1.15)	R180
XCE	IC2	IC2	con-phoenix-508	(2.15 1.425)	R90
XR1	HFBR-2521	HFBR-15X3	fiber-optic-hp	(0.525 0.3)	R0

15.3 APPENDIX 3: MEASUREMENT ELECTRONIC

The measurement electronic circuit, the PCB board were designed by *Helge Lorenzen* using *EAGLE Version 7.5.0* for the project. The componenets and their values along with the circuit are also stored in the SD Card attached to the back of this report.



Measurement Circuit Diagram Designed By Helge Lorenzen for the project.



PCB for the Measurement Electronic Designed By Helge Lorenzen.

15.4 APPENDIX 4: PROGRAM CODE FOR CALIBRATING MEASUREMENT ELECTRONIC (MATLAB CODE)

```
%Program For the Calibration of Measurement Electronic Device For 3
Phase
%Systems
% Developed By: Achema Hosea Egbunu
% Student ID: 142773
clear all;
close all;
clc;

DCInVoltage = 268;                          %DC Input voltage
DCTrue = DCInVoltage* sqrt(2)/sqrt(3);      %Phase voltage: Line-to-
Neutral

%Input the analog measurement values for RST phases:
%RSTRawAnalogVal = [+R, +S, +T; -R, -S, -T],

%Input the measured analog values for R, S, T
RSTRawAnalogVal = [798, 798, 798; 226, 226, 226];

%Change in analog values for each R,S,T Phases
ChangInRSTRawAnalogVal          =          (RSTRawAnalogVal(1,:)          -
RSTRawAnalogVal(2,:));
MeanAnalogVal   =   mean(RSTRawAnalogVal);      %Mean   value   for   the
RSTRawAnalogVal
```

```
%Analog value representing the DCOffset from measurement device
DCOffset = (MeanAnalogVal - 512);

%change (dV) in DCInVoltage: +DCInVoltage - (-DCInVoltage)
ChangDC = 2 * DCInVoltage;

VoltperDigit = ChangDC./ChangInRSTRawAnalogVal  %Slope
MeasurementRange = [(DCOffset + 512).*VoltperDigit; ((DCOffset -
512).*VoltperDigit)];
```

15.5 APPENDIX 5: DETERMINE THE NON-LINEAR BEHAVIOUR FOR THE MEASUREMENT ELECTRONIC (MATLAB CODE)

```
%Purpose: Finding the behaviour of the Measurement Device to the Input
%voltage to determine the range for the linearity of the measurement
device

%Measurement Data After Calibrating the Measurement Electronic Device
for
%the 3 Phases.
clear all;
close all
clc
DCInVoltage = [-300 -250 -200 -150 -100 -50 0 50 100 150 200 250 300];
R = [16 27 74 182 292 401 512 625 734 842 951 992 1004];
S = [16 27 74 182 292 401 512 625 734 842 951 992 1004];
T = [16 27 74 182 292 401 512 625 734 842 951 992 1004];
plot(DCInVoltage, R, '*');
hold on;
plot(DCInVoltage, S, '-.');
hold on;
plot(DCInVoltage, T);
legend('R', 'S', 'T');
xlabel('Voltage');
ylabel('RST Measurements');

%%
%Remark:
%The non-linearity behaviour starts after -200V and +200V
%To Control the voltage in the non-linear range of the plot, code the
%corresponding analog values representing the desired input voltages
in the
%code after: +200V and -200V.
```

15.6 APPENDIX 6: RS232 HARDWARE CIRCUIT

The hardware circuit for the RS232 to Serial Communication designed by Helge Lorenzen for the project.

15.7 APPENDIX 7: INVESTIGATING ANALOGREAD() FUNCTION EXECUTION TIME

```
/*Project Name: Investigating AnalogRead() Execution Time
  Developed By: Achema Egbunu (ID: 142773)*/

int analogPin0 = A0;              //Analog Pin for Reading Analog
Values
//int analogWritPin = 6;          //Pin for Writing Analog Values

/*Creating an Array store the value*/
unsigned long starttimeRd[10];    //Array for storing the start time
for 'AnalogRead()'
unsigned long stoptimeRd[10];     //Array for storing the stop time
for 'AnalogRead()'
unsigned long vals[10];           //Array for storing the
'AnalogRead()' Value

void setup()
{
  /*Setup Serial port and Initialise the pins A0 & pin 6 as Output*/
  Serial.begin(9600);
  pinMode(analogPin0, INPUT);
}

void loop()
{
  int i;
  for (i = 0; i < 10; i++)
  {
    /*Investigating Execution Time for AnalogRead() Function*/
    starttimeRd[i] = micros();
    vals[i] = analogRead(analogPin0);
    stoptimeRd[i] = micros();
  }

  /*Dispaly the Results on Serial Monitor*/
  Serial.println("\n\n---Results---");
  for (i = 0; i < 10; i++)
  {
    /*Displaying the Result for the AnalogRead Values and Exection
     Time*/
    Serial.print("Analog Read Value: ");
    Serial.print(vals[i]);
    Serial.print("  Analog Read Exec Time: ");
    Serial.print(stoptimeRd[i] - starttimeRd[i]);
    Serial.print(" us");
  }
  delay(10000);
}
```

15.8 APPENDIX 8: MAIN PROGRAM CODE

```
/*#######################################################
```

```
  ######          ACHEMA HOSEA EGBUNU              ######
  ######              ID: 142773                   ######
  ######            MASTER THESIS                  ######
  ##########################################################*/

/*################################################################
  #### Defining Libraries for the SPI, Ethernet and SD Card #####
  ################################################################*/
#include <SPI.h>
#include <Ethernet.h>
#include <SD.h>
/*################################################################*/

#define nMux 3   //Number of ADC Channels to be used

/*Output Pin Connected to power electronic
  (Excitation Board) to the Machine */
#define PwmOutPin 3

/*################################################################
  #### DEFINITIONS FOR HARDWARE CONFIGURATIONS FOR ADC,##########
  ####             TIMER0 AND PWM FREQ                ##########
  ################################################################*/

/*#############################################################
  #####          ADC Prescaler Definitions              ######
  ##### For more Information: See Report or ATMEL Manual ######
  #############################################################*/
#define ADCPrescaler2   1
#define ADCPrescaler4   2
#define ADCPrescaler8   3
#define ADCPrescaler16  4
#define ADCPrescaler32  5
#define ADCPrescaler64  6
#define ADCPrescaler128 7

//Definition for enabling: ADC, ADC-Converaion,
//ADC Interrupt and prescalers (ADEN, ADSC, ADIE, & Prescalers)
#define ADCSRAConfig 0xC8 + ADCPrescaler128;

/*#########################################################
  ####   Timer 0 Control Registers (TCCR0A/TCCR0B) ######
  ####       Definitions for Setting Timer         ######
  ####   For More Information: Check Report        ######
  ####           or ATMEL Manuals                  ######
  #########################################################*/

#define WaveformGenerationModeCTC 2 //CTC Mode definition
#define TimerClockDivider1    1 //Division by prescaler of 1(TCCR0B)
#define TimerClockDivider8    2 //Division by prescaler of 8(TCCR0B)
#define TimerClockDivider64   3 //Division by prescaler of 64(TCCR0B)
#define TimerClockDivider256  4 //Division by prescaler of 256(TCCR0B)
#define TimerClockDivider1024 5 //Division by prescaler of
1024(TCCR0B)


/*#########################################################
```

```
  #### Prescalers for Changing PWM Frequency      ######
  #### Remark: Pin 3 is used as PWMOuptut for the ######
  #### control and pin 3 is assigned by default   ######
  #### to timer2, therefore, prescaler for Timer2 ######
  #### definitions are used Base Frequency for    ######
  #### pin 3 is: 31250Hz.                         ######
  #### PWM Freq = Base Freq/Prescaler             ######
  ##################################################*/
#define Divider1    1   //f=31250
#define Divider8    2   //f=31250/8 = 3906
#define Divider32   3   //f=977
#define Divider64   4   //f=488
#define Divider128  5   //f=244
#define Divider256  6   //f=122
#define Divider1024 7   //f=31


/*################################################
  #### Masks for choosing analog input channel #####
  #### For details see ADMUX-Register in the   #####
  #### Report or Atmel manual                  #####
  ################################################*/
/*"ORed" with the wanted Channel number to select a
   channel and remove the rest*/
#define ADMUXConfig  0x40

//"ANDed" with ADMUX to get select Channel number
#define ADMUXChannelsOnly 0x1F

/*-----------------------------------------------------*/
/*#####################################################
  #### Symbolic Access To Global Variables         #####
  #### NOTE: The values for this names are hard    #####
  #### encode in the html-files Any change here    #####
  #### might force an update for every file        #####
  #### involved page!!!                            #####
  #### Advantages: Efficient regarding program     #####
  #### code and performancelog (as block), parse   #####
  #### (compare integers instead of strings)       #####
  #####################################################*/

/*Remark: Idx       = Stands for Index value for the array
          Sym       = Stands for Symbolic Name for the "Idx"
                      value: Idx value can assum other definitions
          gArByte   = Stands for global Array Byte
          gArINT    = global Array Interger
          gArLong   = Stands for global Array Long Interger*/

/*-----------------------------------------------------*/

/*################################################
  #### Symbolic names and values for global     #####
  #### bytes (8bit) values the symbolic name    #####
  #### idx* gives the index of the value in the #####
  #### array where its stored: "gArByte"        #####
  ################################################*/
/*Remark:
```

```
  - definition with 'Idx' represent Array space for storing
    the respective values
  - definition with 'Sym' means the Array space can assume
    other value with the symbol 'Sym'*/
#define IdxlogSemaphore       0   //system logic/timing:
#define SymLogOff             0   //log off mode"
#define SymLogWriteProgress   1   //Main loop write to SD card
#define SymLogGetFirst        2   //ISR forced to reset time axis
#define SymLogGetNext         3   //ISR updates buffered for logging
#define IdxADCRun             1   //system logic/timing for channel
#define IdxWebAnswerMode      2   //Dispaly or Output formats for HMI
#define AMRaw                 0   //Requested file as (txt, htm, raw
binary)
#define AMParsedHTML          1   //Replace placeholders #<placeholder
name>#
#define AMCSV                 2   //Convert (log data) to CSV
#define IdxSetDuty            3   //Setpoint: duty cycle (Open Loop
Control)
#define IdxControlMode        4   //Select control mode: PID, Fuzzy...

#define SymOpenLoopControl    0   //Open loop control
#define SymPidControl         1   //PID control
#define SymFuzzyControl       2   //Fuzzy logic control
#define SymNNControl          3   //Neural Network control (For Future
use)
#define SymTestNewControl     4   //For future use (Scalability)

/* PID-controller parameters */
#define IdxPIDKP              5   //Control coefficient for: (P)-
Control
#define IdxPIDKI              6   //Control coefficient for: PI-
Control
#define IdxPIDKD              7   //Control coefficient for: PID-
Control
#define IdxPWMDuty            8   //Store the value of duty cycle

#define IdxCmd                9   //Command to be executed
#define SymCmdNone            0   //Set after execution other command
#define SymCmdSetControl      1   //Notify that a parameter was
changed
#define SymCmdStartLog        2   //Reqest to open log-file
#define SymCmdStopLog         3   //Request to close log-file on SD-
Card
#define SymCmdStoreDefault    4   //store global variables as default

/* It makes little sense to change the duty cycle
 * with a frequency close to PWM-frequency
   therefore we use a control trigger to count
   down for change of Duty cycle in control.*/
#define IdxControlTriggerActual   10   //Store value for counting down
#define IdxControlTriggerInterval 11   //Set the value for fast
control
#define IdxDebugByte              12   //Debugging Purpose HMI (spy):

/*Size of Array: (index of last entry + 1)*/
#define ngByte                    13  //Size of the global array byte
volatile byte gArByte[ngByte];        //Global Array Byte
```

```
/*#####################################################
  #### Symbolic names and values for global     ######
  #### integer (16bit) values the symbolic      ######
  #### name idx* gives the index of the value   ######
  #### in the array where its stored: "gArINT"  ######
  #####################################################*/
#define IdxR      0   //Sampled instantaneous analog value for phase 1
#define IdxS      1   //Sampled instantaneous analog value for phase 2
#define IdxT      2   //Sampled instantaneous analog value for phase 3
#define IdxBufR   3   //Buffered values of R for logging to SD Card
#define IdxBufS   4   //Buffered values of S for logging to SD Card
#define IdxBufT   5   //Buffered values of T for logging to SD Card

/* Size of Array: (index of last entry + 1)*/
#define ngINT     6     //Size of the Array: "gArInt"
volatile int gArInt[ngINT];  //Global Array


/*#####################################################
  #### Symbolic names and values for global long#####
  #### integer (32bit) values the symbolic name #####
  #### idx* gives the index of the value in the #####
  ####     arraywhere its stored: "gArLong"     #####
  #####################################################*/

/* Alternating position (see adress offset gIdxNewest)
  for the actual and previous computed Measure from
  RST (In ComputeActual Value())*/
#define IdxMeasure          0   //Toggle new and old values measure
//alternated index          1   //Toggle new and old values measure

/* Alternating position (see adress offset gIdxNewest)
   for the actual and previous filtered
   Measure (In ComputeActual Value()) */
#define IdxFilter           2   //Toggle new and old values filter
//alternated index          3   //Toggle new and old values of fileter

#define IdxDMeasure         4   //derivative of the Measure
#define IdxDFilter          5   //derivative of the filtered Measure
#define IdxSetPM            6   //Setpoint with unit as measure
#define IdxDCOffset0        7   //DCOffset value
#define IdxIdle             8   //Counter idle loops outside ISRs
#define IdxLogTime          9   //Time axis to be buffered for logging
#define IdxLogTimeBuf      10   //Time axis for the logging
#define IdxSetMeasure      11   //Setpoint as Measure from HMI
#define IdxErrorSum        12   //store sum of the Errors use Control
#define IdxError           13   //Values from control for debugging
purpose

/*Setpoint to the controller if Voltage is
provided as setpoint from HMI but function
need to be enabled in the mainloop*/
#define IdxSetptVoltToSetPtMeasure 14

/*Array Size: (index of last entry + 1)*/
```

```
#define ngLong              15  //The size Array
volatile long int gArLong[ngLong]; //Global Array Long
/*-------------------------------------------------------*/

#define ADStartConveration 0x40  //ADC Start Conversion bit (ADSC)
const int SDCardPin = 4;      //SD Card communication pin with SPI


/*#####################################################
  #### Toggle between 0 & 1. Uses for Computing #####
  #### the Deviation between New and Old vlaues #####
  #### of: IdxMeasured and IdxFilter           #####
  #####################################################*/
volatile byte gIdxNewest = 0;


/*#####################################################
  #### Defining the Ethernet Settings and Server  #####
  ####          used by Webserver                 #####
  #####################################################*/
  //MAC Address for ARDU. 60HZ M/C:
  //{0x90, 0xA2, 0xDA, 0x10, 0x6C, 0x32};
byte mac[] = {0x90, 0xA2, 0xDA, 0x10, 0x64, 0xE5};
IPAddress ip(192, 168, 0 , 80);
EthernetServer server(80);

File LogFile; /*Filename for Data logging*/
/*-------------------------------------------------*/

/*Assinging the Interrupt Service Routines*/
ISR(ADC_vect);
ISR(TIMER0_COMPA_vect);

/*-------------------------------------------------*/
/*#########################################
  ####             MAIN SETUP          #####
  #### RUNS ONLY AFTER STARTUP OR RESET #####
  #########################################*/
void setup()
{
  /*Disable Global Interrupt to prevent
    Interrupting Setup Initialization*/
  noInterrupts();

  /* Initialize the Serial Com Port
     for Debugging Purpose*/
  Serial.begin(9600);

  /*Wait for serial port to connect.
    Needed for Leonardo (only)*/
  while (!Serial) {
    ;
  }
  Serial.println("...");  //For debugging

  /*Store values of DCOffset Implemented in the
    Measurement Hardware*/
```

```
gArLong[IdxDCOffset0] = 786432; //=3*512*512;

/* ###################################################
   #### Initialization of Timer 0 for real      #####
   #### time application See definitions         #####
   #### above and ATMEL manual for  a detailled #####
   #### description of related registers and    #####
   ####          parameters                     #####
   ###################################################*/
TCCR0A = WaveformGenerationModeCTC;
TCCR0B = TimerClockDivider64;
OCR0A = 255;
/*                                8E6
  interrupt frequency= --------------------------------
                       TimerClockDivider*(1+OCR0A)       */

/*Enable timer0 interrupt*/
TIMSK0 = (1 << OCIE0A);

/* ###################################################
   #### Initialization of ADC-conversion. See  #####
   #### definitions above and ATMEL manual for #####
   #### a detailled  description of related    #####
   #### Registers and parameters               #####
   ###################################################*/
/*                     13*ADCPrescaler
  conversation time = -------------------
                          16E6           */
ADMUX = ADMUXConfig;      //See prescaler options defined above
ADCSRA =  ADCSRAConfig;
ADCSRB =  0;

/*################################################
  #### Drive the Output pin high to Invert  #####
  #### the signal on the Opto Device on the #####
  #### Excitation Board to 0                #####
  #### (Due to Inversion of Opto Device)    #####
  ################################################*/
analogWrite(PwmOutPin, 255);

/*SD-Card Initialization*/
if (SD.begin(SDCardPin))
{
  Serial.println("SD:Ok");
}
else
{
  Serial.println("SD: Error");
}

/*Initialize Ethernet and Server*/
Ethernet.begin(mac, ip);
server.begin();

/*#########################################################
  #### A Calling a Function that changes            #####
  #### PWM Frequency of the Output pin PWMOutPin: 3 #####
```

```
    ######################################################*/
  ChangePWMFrequency(Divider32);

  /*Start Global Interrupt*/
  interrupts();
}


/* ###############################################
   ###                MAIN LOOP                ####
   ### Executing Costly and Time Consumming ####
   ###    Actions with Lowest Priories       ####
   ###############################################*/

void loop()
{
  /*###############################################
    #### Variables and Pointers for Logging #####
    ####          Buffered Values            #####
    ###############################################*/
  long aLong;
  long *pLong = &aLong;
  int aInt;
  int *pInt = &aInt;
  byte aByte;
  byte *pByte = &aByte;

  if (gArByte[IdxlogSemaphore] <= SymLogWriteProgress)
  {
    /* ####################################################
       #### Execute this Function because it has     #####
       #### the sole Responsibility to Initiates     #####
       #### logging of buffered values to SD Card.   #####
       #### Webserver may do every thing including   #####
       #### opening or close log-File and also has   #####
       #### to ensure itself suitable log-semaphore  #####
       ####          values on leave               #####
       ####################################################*/
    DoWeb();

    /*#############################################################
      #### Uncomment this Function If Setpoint From HMI    #####
      #### (User Interface) is in Voltage (Phase-to-Phase)#####
      #############################################################*/
    //ComputeVoltToSetpoint(gArLong[IdxSetMeasure]);

    if (gArByte[IdxlogSemaphore] == SymLogWriteProgress)
    {
      /*###################################################
        #### Log the Raw Values of RST Phases to File #####
        ###################################################*/
      aLong = gArLong[IdxLogTimeBuf]; LogFile.write((byte *) pLong,
4);
      aInt = gArInt[IdxBufR]; LogFile.write((byte *) pInt, 2);
      aInt = gArInt[IdxBufS]; LogFile.write((byte *) pInt, 2);
      aInt = gArInt[IdxBufT]; LogFile.write((byte *) pInt, 2);
```

```
      /*#############################################################
        #### Set Semaphore to Get Next Updated Values From ISR ####
        #############################################################*/
      gArByte[IdxlogSemaphore] = SymLogGetNext;
    }
  }
  else
  {
    /*###############################################
      #### Increment the Idle Counter Gives   #####
      #### Information of Timing Control and  #####
      ####         Processor usage           #####
      ###############################################*/
    gArLong[IdxIdle]++;
  }
}


/*###########################################
  #### FUNCTION TO CHANGE PWM FREQUENCY #####
  #### INPUT: PRESCALER VALUE           #####
  #### TIMER 2 USED BECAUSE PIN 3       #####
  #### and 11 ARE ASSIGNED TO IT        #####
  #### BY DEFAULT AND PIN 3 (PWMOutPin) #####
  #### USED FOR CONTROL                 #####
  ###########################################*/
void ChangePWMFrequency(byte newDivider)
{
  if (newDivider <= Divider1024)
  {
    TCCR2B = (TCCR2B & 0b11111000) | newDivider;  //TCCR2B Contains
  }
}


/*#########################################################
  #### FUNCTION FOR COMPUTING CORRESPONDING ANALOG #####
  #### VALUES FOR A SETPOINT GIVEN IN VOLTAGE      #####
  ####              FROM THE HMI                   #####
  #### REMARK: THIS INCREASES THE PROCESS LOAD     #####
  ####            DUE TO FLOAT COMPUTATIONS        #####
  #########################################################*/
void ComputeVoltToSetpoint(long int voltSetpt)
{

#define Upp         0
#define AngleR      1
#define AngleS      2
#define AngleT      3
#define measureR    4
#define measureS    5
#define measureT    6
#define factor      7

#define nDouble     8      //Array size
double LArDouble[nDouble]; //Array for double
```

```
#define adcValueR   0
#define adcValueS   1
#define adcValueT   2
#define setpointVal 3

#define nINT 4           //Array size
long int LArINT[nINT];   //Array for Interger

  /*#########################################################
    ###### Scaling Factor for AC 3P Voltage to Analog ####
    ####      Value based on Measurement hardware used####
    #### Remark: @ 200Vac = 951;                       ####
    ####         @ 0Vac   = 512;                       ####
    ####         @ -200V  = 74 (Analog values)         ####
    ####                                               ####
    ####              439                              ####
    #### adcValue =[------.Upp] + 512;                 ####
    ####              200                              ####
    #### Upp = phase-to-phase voltage for the machine ####
    #########################################################*/
    //setpointmeasure = [adcValueR^2+adcValueS^2+adcValueT^2]-DCOffset

  /*#########################################################
    ### Setting a Limit to the phase-to-phase voltage #####
    ###               given by the user               #####
    #########################################################*/
  if (voltSetpt >= 253)
  {
    voltSetpt = 253;
  }


  /*##### Compute the Angle for the each RST Phases ############*/
  LArDouble[Upp] = 0.8165 * voltSetpt; //Peak voltage:
(VLL*sqrt(2))/sqrt(3)
  LArDouble[AngleR] = 0;               //Angle: Sin(0)    (degrees)
  LArDouble[AngleS] = -0.8660;         //Angle: Sin(0 - 120)(degrees)
  LArDouble[AngleT] =  0.8660;         //Angle: Sin(0 - 240)(degrees)


  /*##### Compute the Amplitude for each RST Phases ##############
    Measure of R phase: Vp*Sin(0)
    Measure of S phase: Vp*Sin(0-120)
    Measure of T phase: Vp*Sin(0-240)*/

  LArDouble[measureR] = (LArDouble[Upp]) * (LArDouble[AngleR]);
  LArDouble[measureS] = (LArDouble[Upp]) * (LArDouble[AngleS]);
  LArDouble[measureT] = (LArDouble[Upp]) * (LArDouble[AngleT]);

  /*############ Compute Analog Values for each RST Phases
###############*/
  LArDouble[factor] = 2.195;           //Scaling factor: factor =
439/200
  LArINT[adcValueR] = ((LArDouble[factor]) * (LArDouble[measureR])) +
512;
  LArINT[adcValueS] = ((LArDouble[factor]) * (LArDouble[measureS])) +
512;
```

```
  LArINT[adcValueT] = ((LArDouble[factor]) * (LArDouble[measureT])) +
512;

  /*### Compute the Corresponding Measured Value for Voltage Setpoint
######*/
  gArLong[IdxSetptVoltToSetPtMeasure] =
              (((LArINT[adcValueR]) * (LArINT[adcValueR]) //R * R
            + (LArINT[adcValueS]) * (LArINT[adcValueS])   //S * S
            + (LArINT[adcValueT]) * (LArINT[adcValueT]))  //T * T
            - (gArLong[IdxDCOffset0]));               //Remove
DCOffset
}
```

15.9 APPENDIX 9: CODE FOR ADC_ISR (SAMPLING DATA FROM ANALOG PINS)

ADC_ISR Code:

```
/*#########################################################
  ######          ACHEMA HOSEA EGBUNU              ######
  ######              ID: 142773                   ######
  ######            MASTER THESIS                  ######
  #########################################################*/

/*#############################################################
  ##### This Function is Called after a Conversion of   #####
  ##### every analog sample, Scan the analog channels,  #####
  ##### Executes Computation of actual measured values, #####
  ##### control algorithm, Buffer values for logging.   #####
  ##### Remark: All lower priority tasks such as        #####
  #####         webserver are executed between the      #####
  #####         Interrupts in the main loop.            #####
  ##### NOTE: Not advisable to do Comport operation     #####
  #####       like: Serial print due to timing control  #####
  #############################################################*/

ISR(ADC_vect)
{
#define LastRun nMux-1
  /*For fetching analog sample from the ADC Register before
    next conversion*/
  unsigned int AnaBufL, AnaBufH;
  byte MUXPos;                //Temporary keep MUX Channel

  //Fetch data from lower and higher byte of teh ADC Register
  AnaBufL = ADCL; AnaBufH = ADCH;

  /*#########################################################
    ###### Start parallel conversation when needed  ######
    #########################################################*/
  if (gArByte[IdxADCRun] < LastRun)
  {
    //Start next conversion (Note: Conversion is done by hardware)
```

```
  ADCSRA |= ADStartConveration;
}

//Determine the MUX channel to be set next and mask out the Rest
MUXPos = ADMUX & ADMUXChannelsOnly;

//Limit the next MUX channel to maximum number of channels
if (MUXPos < nMux)
{
  MUXPos++;
}
else
{
  MUXPos = 0;
}

/*############################################################
  ##### Store ADC-Value to coresponding array position ######
  ##### Index IdxR,IdxS,IdxT are defined as 0..2 to    ######
  ##### Corresponds to ADCRun                          ######
  ############################################################*/
//Store sampled values into the array
gArInt[gArByte[IdxADCRun]] = (AnaBufH << 8) | AnaBufL;

/*############################################################
  #### Set AD to next MUX channel if all channels   ######
  ####          have not been sampled               ######
  ############################################################*/
if (gArByte[IdxADCRun] < LastRun)
{
  /*Switch next MUX physically while hardware
    conversion is in progress*/
  ADMUX = ADMUXConfig | MUXPos;

  //Determine the loop number for next ISR-call
  gArByte[IdxADCRun]++;
}
else                    //All Channels are sampled
{
  //Set MUX Channel to 0 for next timer interrupt
  ADMUX = ADMUXConfig;

  /*Call the function to compute the measured values
    for the sampled data*/
  ComputeActualValue();

  /*Set PWM for excitation to control generator output voltage*/
  DoControl();

  /*################################################
    ##### Once the main job is done, then:     ######
    ##### Synchronize data logging between     ######
    ##### processes  with logSemaphore         ######
    ################################################*/
  if (gArByte[IdxlogSemaphore] == SymLogGetFirst)
  {
    //Reset time axis for data logging
```

```
      gArLong[IdxLogTime] = 0;
    }
    if (gArByte[IdxlogSemaphore] >= SymLogGetFirst)
    {
      /*Update the log Buffer with values for the
        main loop to log data*/
      /*Update log time buffer with new values,
        consistent with LogValues*/
      gArLong[IdxLogTimeBuf] = gArLong[IdxLogTime];
      gArLong[IdxLogTime]++;       //Increment the log time

      //Update the R, S, T Buffer with new R,S,T values
      gArInt[IdxBufR] = gArInt[IdxR];
      gArInt[IdxBufS] = gArInt[IdxS];
      gArInt[IdxBufT] = gArInt[IdxT];

      /*Reset the array for logSemaphore to enables the
        main loop to write buffered values to SD-Card*/
      gArByte[IdxlogSemaphore] = SymLogWriteProgress;
    }
  }
}
```

15.10 APPENDIX 10: CODE FOR COMPUTING ACTUAL VALUES OF RST

Code for Computing Actual Values:

```
/*############################################################
  ######          ACHEMA HOSEA EGBUNU              ######
  ######              ID: 142773                   ######
  ######            MASTER THESIS                  ######
  ############################################################*/

/*#################################################################
  #######  COMPUTATION OF ACTUAL VALUES AND IIR FILTER   ########
  #######  THE FUNCTION IS CALLED AFTER EVERY COMPLETE   ########
  #######  SETS (RST) OF SAMPLED VALUES TO PROVIDE THE   ########
  #######  DATA USED FOR DIFFERENT CONTROL FUNCTIONS     ########
  #################################################################*/
void ComputeActualValue()
{
    #define dypower 2     //Increase to smooth filtered value
    #define dytrunc (1<<dypower)-1 //Use to compensate integer
truncation
    byte idxold, idxnew;           //Store new and old value of
'gIdxNewest'

    //Toggle index for derivatives Computations used for PID Control
    idxold = gIdxNewest;        //Store the old value of 'gIdxNewest'
    gIdxNewest ^= 1;            //Toggle the global variable and
    idxnew = gIdxNewest;        //Store the newest value to 'idxnew'
```

```
  /*#################################################################
    #######  NOTE: The cast is required for computation    #######
    #######         of long intergerssince truncation of   #######
    #######         interger overflow by C compiler        #######
    ################################################################*/

  /*Square each phase, sum up then remove DCOffset to get a
    measure for control */
  //##########   Compute actual measure  ##############
    gArLong[IdxMeasure+idxnew] = (
      ((long int)gArInt[IdxR] * (long int)gArInt[IdxR])
    + ((long int)gArInt[IdxS] * (long int)gArInt[IdxS])
    + ((long int)gArInt[IdxT] * (long int)gArInt[IdxT]))
    - (long int)gArLong[IdxDCOffset0];

   /*#########################################################
     ##########  Filter Measure using IIR Filter   ##########
     ########################################################*/
    gArLong[IdxFilter + idxnew] =
    (((((long int)gArLong[IdxFilter+idxold])<<dypower)
    + (long int)dytrunc                          //compensate integer
truncation
    - (long int)gArLong[IdxFilter+idxold]    // 2^dypower-1
    + (long int)gArLong[IdxMeasure+idxnew])  //weight actual
    >>dypower);                              //rescale

/*#########################################################################
  ####### Compute Derivatives for Measure and Filtered Values ########

 ########################################################################*
/
     gArLong[IdxDMeasure] =
       (long int)gArLong[IdxMeasure+idxnew]
     - (long int)gArLong[IdxMeasure+idxold];
     gArLong[IdxDFilter] =
        (long int)gArLong[IdxFilter+idxnew]
      - (long int)gArLong[IdxFilter+idxold];
}
```

15.11 APPENDIX 11: CODES FOR P, PI, PID CONTROL, 'CHANGEDUTY()' AND 'DOCONTROL()' FUNCTIONS

```
/*#########################################################
  ######          ACHEMA HOSEA EGBUNU              ######
  ######              ID: 142773                   ######
  ######            MASTER THESIS                  ######
  #########################################################*/

/*-----------------------------------------------------------------------
---*/
/*#################################################################
  #######                    MAIN FRAM TO:                  #######
  #######        CHANGE DUTYCYCLE, SELECT CONTROL MODE,     #######
```

```
  #######      CONTROL FUNCTIONS DEPENDING ON THE SYMBOLIC  #######
  #######          VALUES IN GLOBAL ARRAY WITH             #######
  #######             index: IdxControlMode                #######
  #######      DEVELOPED BY: ACHEMA EGBUNU (ID:142773)     #######
  ######################################################################*/
/*---------------------------------------------------------------------
----*/

/*#########################################################
  ###########      CHANGE DUTYCYCLE           ###########
  ########### FUNCTION TO CHANGE DUTY CYCLE   ###########
  #########################################################*/
void ChangeDuty(int newDuty)
{
  /*Seeting a bond on the Duty Cycle to limit the control
    signal within the PWM range (0<=pwm<=255)*/
  if (newDuty >= 255)
  {
    newDuty = 255;
  }
  if (newDuty <= 0)
  {
    newDuty = 0;
  }
  /*Write to PWMOutPin and the array register only
    if there is changes*/
  if (gArByte[IdxPWMDuty] != newDuty)
  {
    //Store the new pwm to array to monitor on HMI
    gArByte[IdxPWMDuty] = newDuty;

    /*Control output is inverted due to the inversion
     of the opto signal on the excitation board*/
    analogWrite(PwmOutPin, ~newDuty);
  }
}

/*#############################################################
  ######                  DO CONTROL                     ######
  ###### FUNCTION TO SWITCH DIFFERENT CONTROL MODES:     ######
  ###### OPEN LOOP; PID CONTROL; FUZZY LOGIC & NEURAL    ######
  ###### NETWORKS SET IdxControlTriggerActual from HMI   ######
  ######     (System Status) for fast control          ######
  #############################################################*/
void DoControl()
{
  gArByte[IdxControlTriggerActual]--;
  if (gArByte[IdxControlTriggerActual] == 0)
  {
    gArByte[IdxControlTriggerActual] =
gArByte[IdxControlTriggerInterval];
    switch (gArByte[IdxControlMode])
    {
      case SymOpenLoopControl :
        {
          ChangeDuty(gArByte[IdxSetDuty]);
          break;
```

```
        }
      case SymPidControl :
        {
          PID_Control();
          break;
        }
      case SymFuzzyControl :
        {
          ;
          break;
        }
      case SymNNControl :
        {
          ;
          break;
        }
      case SymTestNewControl :
        {
          //TestNewControl();
          break;
        }
        // restart count down
    }
  }
}


/*#########################################################
  #########  (P)(I)(D) CONTROL FUNCTION       ############
  #########   P-CONTROL:   KI = KD = 0         ############
  #########      PI-CONTROL:  KD = 0           ############
  #########  PID-CONTROL: NONE SET TO ZERO     ############
  #########################################################*/
void PID_Control()
{
  long int error;     //Use to store the values of error
  int u;              //Use to store the value of Control Signal
  long int P;         //Proportional part
  long int P_scaled;  //Use for division by bit operation

  long int I;         //Use to store the value of the Integral part
  long int I_scaled;  //Use for division by bit operation
  long int D;         //Derivative part
  long int D_scaled;  //Use for division by bit operation
  int f = 976;        //Frequency: 1/dt, Part of the shifting already
  long int AntiWindup = 31900; //Use for implementing Anti winddup

  /*#############################################################
    ########          COMPUTING THE ERREOR            ##########
    ########            CONTROLLER INPUT:             ##########
    ######## ERROR = SETPOINT - CURRENT FILTERED VALUE ##########
    #############################################################*/
  error = gArLong[IdxSetMeasure] - gArLong[IdxFilter + gIdxNewest];

  /*#############################################################
    #########  SUMMING THE ERRORS FOR INTEGRAL PART   ##########
    ##########          OF THE CONTROLLER             ##########
```

```
  ############################################################*/
gArLong[IdxErrorSum] += error;
gArLong[IdxError] = error;    //For debugging purpose

/*############################################################
 * ########               ANTIWINDUP               #########

  ######### Implementing Anti-Windup to prevent     #########
  ######### the control signal from being saturated #########
  ############################################################*/
if (gArLong[IdxErrorSum] > AntiWindup)
{
  gArLong[IdxErrorSum] = AntiWindup;
}
else if (gArLong[IdxErrorSum] < - AntiWindup)
{
  gArLong[IdxErrorSum] = -AntiWindup;
}

/*############################################################
  ############           P - CONTROL                 #########
  ############ The P Controller; scaled by a factor  #########
  ############       to enhance scaling of Kp.       #########
  ############          Factor: 23; KP: 136          #########
  ############################################################*/

/*Bit shift operation with negative values can leads to
  undefine problem, therefore, this is taken care of here*/
  P_scaled = (gArByte[IdxPIDKP] * error);
  if(P_scaled >= 0)
  {
    P = (P_scaled >> 23);
    }
    else
    {
      P = -(-P_scaled >> 23);
      }

/* ############################################################
  ##########            PI - CONTROL               ##########
  ##########   Integral part, scaled by a factor   ##########

  ##########       to enhance scaling of Ki.       ##########
  ##########          Factor: 22; KI: 131          ##########
  ############################################################*/

/*Bit shift operation with negative values can leads to
  undefine problem, therefore, this is taken care of here*/
I_scaled = (gArByte[IdxPIDKI] * gArLong[IdxErrorSum]);
if(I_scaled >= 0)
{
  I = (I_scaled >> 22);  //Capacitor: KI= 1, shift: >>16 or >>17
  }
  else
  {
    I = -(-I_scaled >> 22); //Hardcord the shift
    }
```

```
  /*#########################################################
   *#######          PID - CONTROL                    #######
    ####### The Derivative part, scaled by a          #######
    ####### factor to enhance scaling of KdComputing  #######
    ####### rate of change in error with time, scaled #######
    ####### by a factor.  Computing the derivative    #######
    ####### using the current and the last output     #######
    #######          Factor: 22; KD: 176              #######
    #########################################################*/




  /*Bit shift operation with negative values can leads to
    undefine problem, therefore, this is taken care of here*/
  D_scaled  = (gArByte[IdxPIDKD] * gArLong[IdxDFilter]);
  if(D_scaled >= 0)
  {
    D = (D_scaled >> 22);
 }
  else
  {
    D = -(-D_scaled >> 22);
    }

  /*#############################################################
    #########    COMPUTING THE CONTROL SIGNAL           #########
    #########       ADDING THE P + I - D                #########
    #########  The -D is because output measurement is  #########
    #########    used to compute the change in error    #########
    #############################################################*/
  u = (long int)P + (long int)I - (long int)D;



   /*#############################################################
    #######     ADDING THE CONTROL SIGNAL TO DUTY CYCLE     #########
    ######## Calling the Function that Writes the Control   #########
    ########              Signal to PWMOutPin               #########
    ######## INPUT TO ARGUMENT: PREV. DUTYCYCLE + CONTROL u #########


    #############################################################*/
  ChangeDuty(gArByte[IdxPWMDuty] + u);
}
```

15.12 APPENDIX 12: CODES FOR 'OPENWRITELOGFILE()' , 'CLOSELOGFILE()', 'UPLOADCSV()' FUNCTIONS

```
/*#########################################################
  ######          ACHEMA HOSEA EGBUNU              ######
```

```
  ######              ID: 142773                    ######
  ######           MASTER THESIS                    ######
  ##########################################################*/

/*-------------------------------------------------------------------*/
/*###################################################################
  #########  FUNCTION FOR ANSWER MODE CSV FORMAT       #########
  #########     CALLED IN THE DOWEB() FUNCTION         #########
  ######### CONVERT THE RESPONSE TO CSV WHILE SENDING  #########
  #########  THE WEBAPPLIED TO FORMAT FOR DOWNLOADING  #########
  #########                  LOG DATA                  #########
  ###################################################################*/
/*-------------------------------------------------------------------*/
  #define LogFileName "log.dat"  //Filename for data logging

void uploadCSV(EthernetClient client, String Filename)
{
  //for holding the read values from file
  long aLong; long *pLong; pLong = &aLong;
  int aInt; int *pInt; pInt = &aInt;
  long Rest;
  byte Col;

  File RawFile = SD.open(Filename);
  if (RawFile)      //if filename is available then
  {
    Col = 0;
    Rest = RawFile.available();
    while (Rest > 1)  //Rest = 1 would stay endless
    {
      switch (Col)
      {
        case 0:
          {
            //Read First Column, the time axis
            RawFile.readBytes((byte *) pLong, 4);
            client.print(aLong); client.print(','); Col++; break;
          }
        case 3:
          {
            RawFile.readBytes((byte *)pInt, 2);    //Last Column
            client.println(aInt); Col = 0; break;  // between first
and Last
          }
        default:
          {
            RawFile.readBytes((byte *) pInt, 2);
            client.print(aInt); client.print(','); Col++; break;
          }
      }
      Rest = RawFile.available();
    }
    RawFile.close();
  }
}
```

```
/*#####################################################################
  #########  FUNCTION TO OPEN LOG FILE FOR LOGGING AND   #########
  #########  SET SEMAPHORE THE FILENAME IS HARD ENCORDED #########
  #####################################################################*/
void OpenWriteLogFile()
{
  if (SD.exists(LogFileName))
  {
    SD.remove(LogFileName);
  }
  LogFile = SD.open(LogFileName, FILE_WRITE);
```

```
  /*#####################################################################
    ########  Main loop can log data to opened file.        ########
    ########  change of semaphore informs ISR to buffer     ########
    ########  the updated values of RST. The ISR also       ########
    ########  changes the semaphore which causes the        ########
    ########  buffered data to be written to the file       ########
    ########  from the main loop.                           ########
    ########  NOTE: It may happen that some samples         ########
    ########  are not written to file due to execution      ########
    ########  higher priorities functions                   ########
    ######## (computeactual values & control) by interrupt.########
    ######## Therefore the number of ISR-call has to be     ########
    ######## logged to get a coresponding time axis for     ########
    ######## the samples                                    ########
    #####################################################################*/
  if (LogFile)
  {
    //Forces the time axis to reset to zero in ISR
    gArByte[IdxlogSemaphore] = SymLogGetFirst;
  }
  else
  {
    gArByte[IdxlogSemaphore] = SymLogOff; //Do nothing
  }
}
```

```
/*#####################################################################
  ############ FUNCTION TO STOP LOGGING AND CLOSE FILE  ############
  #####################################################################*/
void CloseLogFile()
{
  //Check if the file is opened
  if (gArByte[IdxlogSemaphore] > SymLogOff)
  {
    LogFile.close();                      //Then close the file
    gArByte[IdxlogSemaphore] = SymLogOff; //Stop any action for
datalogging
```

```
  }
}
```

15.13 APPENDIX 13: CODE FOR THE WEB SERVER AND ITS SUBFUNCTIONS

```
/*#########################################################
  ######          ACHEMA HOSEA EGBUNU              ######
  ######              ID: 142773                   ######
  ######            MASTER THESIS                  ######
  #########################################################*/
/* ============================================
  Web Server for the 60Hz syncronous machine

  ============================================ */
#define DefaultPage "index.htm"

//limit length of string to avoid stack overrun
#define maxLengthRequest 50
#define MaxParseLength   10

/* String to integer for multi purpose use */
long int getValFromString(String w, int p)
{
  byte loop = p;
  String NumStr = ""; //Start with empty string

  //while the digit value (0..9) is found then
  while (isDigit(w[loop]))
  {
    NumStr.concat(w[loop]);   //Build a number string

    /*increment position to point to the next
      character in the actua line*/
    loop++;
  }
  return (NumStr.toInt()); //Return the digit value
}

/*################################################
  ### FUNCTION TO PRINT IP ADDRESS OF ARDUINO ###
  ###         CONNECTED TO THE MACHINE         ###
  ################################################*/
String DisplayAddress(IPAddress address)
{ return String(address[0]) + "." +
         String(address[1]) + "." +
         String(address[2]) + "." +
         String(address[3]);
}

/* ################################################
   ### Just answer with requested file as it is #####
   ################################################*/
void AnswerFile(EthernetClient client, String Filename)
{
```

```
  char c;
  File AnwerFile = SD.open(Filename);
  if (AnwerFile)  //file available
  {
    while ((AnwerFile.available()) && client.connected())
    {
      //Read the file and send to the client (web)
      c = AnwerFile.read(); client.write(c);
    }
    AnwerFile.close(); //Close file after all sent to web
  }
}
```

```
/* ###################################################
   #### FUNCTION TO REPLACE THE PLACEHOLDERS    #####
   ####       (format: #<ph>#) In STRING        #####
   ###################################################*/


String ReplacementString(String s1)
{
  if (s1 == "IP")
  {
    /*If the placeholder is 'IP' replace it with
      IP address*/
    return DisplayAddress(Ethernet.localIP());
  }
  if (s1.startsWith("gB", 0))
  {
    /*if the placeholder starts with 'gB', call the
      function to return the digit value (Idx value)
      and replace the string 'gBIdx' with the content
      of the array 'gArByte[]' representing the
      index: Idx. that is: Repalce: gArByte[Idx] with
      gBIdx while responding to the client*/
    return String(gArByte[getValFromString(s1, 2)]);
  }
  if (s1.startsWith("gI", 0))
  {
    //Same approach as explained above
    return String(gArInt[getValFromString(s1, 2)]);
  }
  if (s1.startsWith("gL", 0))
  {
    //Same approach as explained above
    return String(gArLong[getValFromString(s1, 2)]);
  }
  return ("???"); //return "???" if no placeholder is found
}

/* ###################################################
```

```
   #####         ANSWER WITH PARSE HTML FILE          #####
   ##########################################################*/
void AnswerFileParsed(EthernetClient client, String Filename)
{
  char c;
  String s1 = "";      //Start with empty string

  File AnwerFile = SD.open(Filename);
  if (AnwerFile)               //File available
  {
    while ((AnwerFile.available()) && (client.connected()))
    {
      c = AnwerFile.read();    //Read teh file
      if (c != '#')
      {
        client.write(c);     //Write to web
      }
      else                     //Found a placeholder
      {
        c = AnwerFile.read(); //Read next character
        s1 = "";              //new  replacement identifier
        while ((c != '#') && (client.connected()) && (s1.length() <
           MaxParseLength))
        {
          s1.concat(c);         //Append char to string
          c = AnwerFile.read(); //Read next character
        }

        /*Parse string to replacement placeholder function
          to replace the placeholder with values in the array
          representing their Idx and print the return value to
          web (client)*/
        client.print(ReplacementString(s1));
      }
    }
    AnwerFile.close(); //Close file after response to client
  }
}

/*#################################################################
  #### FUNCTION TO SEARCH FILENAME AND Idx FROM THE REQUEST   ####
  ####                  STRING FROM THE CLIENT                ####
  ####  AND RETURN FILENAME AND STORE THE VALUE FOR Idx TO    ####
  ####                     RESPECTIVE ARRAY                   ####
  #################################################################*/
String NewParseRequest(String GetString)
{
  String aFileName;

  //Find position in the string with question mark
  int p = GetString.indexOf("?");
  int idx;          //array index of value
  long v;
  if (p > 0)
  {
    /*If position is found then search from position 5 to
       position 'p' to extract the file name*/
```

```
    aFileName = GetString.substring(5, p); //5=lengthOf("GET /"
    while (p > 0)
    {
      GetString.remove(0, p + 1);    /*also the ?*/
      idx = getValFromString(GetString, 2); /*length gB,gI,gL always
2*/

      //Find the corresponding value for the Idx
      v = getValFromString(GetString, GetString.indexOf("=") + 1);

      if (GetString.startsWith("gB", 0))
      {
        gArByte[idx] = v;  //Store value of gB in array
      }
      if (GetString.startsWith("gI", 0))
      {
        gArInt[idx] = v;
      }
      if (GetString.startsWith("gL", 0))
      {
        gArLong[idx] = v;
      }
      p = GetString.indexOf("&");
    }
  }  /*additional parameters*/
  else  //if file name does not end with "?" (p not available) then
  {
    //Filename ends before " HTTP/"
    aFileName = GetString.substring(5, GetString.lastIndexOf(' '));
  }
  if (!SD.exists(aFileName))
  {
    //Use default if filename does not exist
    aFileName = DefaultPage;
  }
  return (aFileName);  //Return filename
}

/* ###################################################
   ######   FUNCTION TO ANALYZE CLIENT REQUEST ######
   ###################################################*/
String GetRequest(EthernetClient client)
{
  char c;
  String s1 = "";   //empty the string
  String Filename = DefaultPage;
  while (client.connected() && client.available())
  {
    //character available from the client
    c = client.read();          //Read the request
    if (c == '\n')              //Reached end of line
    {
      if (s1.length() != 0)
      {
        if (s1.startsWith("GET /", 0))
        {
          /*Parse string to function responsible for
```

```
            extracting the filename and extract value for Idx
            and store in the array*/
          Filename = NewParseRequest(s1);
        }
      }
      s1 = "";
    }   //Clear to start with next row
    else
    { /*If end of line not reach, append character to string
        to build request line*/
        //limit length to avoid stack overrun
      if ((c != '\r')&& (s1.length() < maxLengthRequest))
      {
        s1.concat(c);
      }
    }
  }
  return (Filename); // note: do not return until buffer is empty
}


/* ################################################
   ###  Main Function to handle web requests ######
   ################################################*/
void DoWeb()
{
  //Listen for incoming clients
   //Create a client instance for each web request
  EthernetClient client = server.available();

  if (client)     //Client available
  {
    gArByte[IdxWebAnswerMode] = AMRaw;   //default raw, as it is

    //Get requested file and Parameters (stored in global arrays)
    String Filename = GetRequest(client);

    /*Operating in Open Loop Control */
    if (gArByte[IdxControlMode] == SymOpenLoopControl)
    {
      /*Call the function responsible for writing the duty
        cycle to pwm pin */
      ChangeDuty(gArByte[IdxSetDuty]);
    }

    /*#################################################################
      #### Execute Actions as requested as Parameters from HMI ####
      #################################################################*/
    switch (gArByte[IdxCmd])
    {
      case SymCmdStartLog :     //log semaphore is handled, too
        {
          //Logging mode, function to open file for logging is called
          OpenWriteLogFile();
          break;
        }
      case SymCmdStopLog :       //log semaphore is handled, too
```

```
        {
          //Stop logging mode, function for closing log file is called
          CloseLogFile();
          break;
        }
    }
    gArByte[IdxCmd] = SymCmdNone; //Clear the already executed command

    /* ################################################
       ####            Upload Answer (file)        ####
       ################################################ */
    switch (gArByte[IdxWebAnswerMode])
    {
      case AMCSV :
        {
          //Uploading CSV Format as response to client
          uploadCSV(client, Filename);
          break;
        }
      case AMParsedHTML :
        {
          //Uploading Response as HTML Format to the client
          AnswerFileParsed(client, Filename);
          break;
        }
      default :
        {
          //Response to client as raw data
          AnswerFile(client, Filename);
          break;
        }
    }
    //Stop the client after response to request
    client.stop();
  }
}
```

15.14 APPENDIX 14: HTML FILES

15.14.1 HTML MAIN DESIGN-FILENAME: MAIN0.HTM

<base target="_self">

HTTP/1.1 200 OK

Content-Type: text/html

Content-Location: /index.htm

Connection: close

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
<meta name="author" content="Achema Egbunu">
<title>PCC60HzIntro</title>

</head>
<!-- Note:
  Text bracketed in hash character is interpreted as placeholder.
  See examples below.
  Therefore: Never use hash character in files of this project
  exept to define placeholders !!!  -->
<body>

<h1>60Hz-Exitation Control</h1>
<p>you are connected to Arduino for 60Hz voltage control at IP:#IP#</p>
<p>Harware: V2.1<br>Software: V.0.1
</p>
<h2>Variable Voltage (Open- or Closed loop Control; phase2phase 120VAC..250VAC)</h2>
<p><img alt="external reference" src="http://www.thebox.myzen.co.uk/Tutorial/Media/PWMan.gif">Image
embedded from: http://www.thebox.myzen.co.uk/Tutorial/Media/PWMan.gif</p>
<h2>Fixed frequency=60Hz by
mechanical transmission</h2>
```

<img alt="external reference" src="http://i59.tinypic.com/4sz32v.gif">Image embedded from: http://i59.tinypic.com/4sz32v.gif

</body>

</html>

15.14.2 HTML TITLE DESIGN-FILENAME: TITLE.HTM

HTTP/1.1 200 OK

Content-Type: text/html

Content-Location: /index.htm

Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">

<meta content="de" http-equiv="Content-Language">

<meta http-equiv="expires" content="0">

<meta name="author" content="Achema Egbunu">

<title>60HzPCCBanner</title>

<base target="_self">

</head>

<body>

<h1>60Hz-PCC</h1>

```
</body>

</html>
```

15.14.3 HTML STANDARD MENU-FILENAME: INDEX.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /index.htm
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
<meta name="author" content="Achema Egbunu">
<title>60HzPCCBanner</title>
<base target="_self">
</head>

<body>

<h1>60Hz-PCC</h1>

</body>
```

```
</html>
```

15.14.4 HTML SETPOINT-FILENAME: SETPTS.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /index.htm
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
<meta name="author" content="Achema Egbunu">

<title>PCC60HzSetpoints</title>

</head>
<!-- Note:
  Text bracketed in hash character is interpreted as placeholder.
  See examples below.
  Therefore: Never use hash character in files of this project
  exept to define placeholders !!!  -->
<body>

<table border="2">
        <tr>
```

```
            <td style="height: 23px">Setpoint Duty Cycle (0:255)<br>for open loop
            control</td>
            <td style="height: 23px">Setpoint Measure<br>for close loop
            control</td>
        </tr>
        <tr>
            <td>
<form action="SetPts.htm" method="get" style="width: 150px; height: 22px;" title="SetPoint">
        <input name="gB3" type="text" value="#gB3#" />
         <input type="hidden" name="gB2" value="1">
         <input type="hidden" name="gB4" value="0">
        </form>
            </td>
            <td>
<form action="SetPts.htm" method="get" style="width: 150px" title="SetPoint">
        <input name="gL11" type="text" value="#gL11#" />
         <input type="hidden" name="gB2" value="1">
         <input type="hidden" name="gB4" value="1">
         </form>
            </td>
        </tr>
        <tr>
            <td>
             </td>
            <td>
            Measure, <br/>take as setpoint<form action="SetPts.htm" method="get"
style="width: 150px" title="SetPoint">
        <input name="gL11" type="text" value="#gL0#">
```

```
        <input type="hidden" name="gB2" value="1">
    <input type="hidden" name="gB4" value="1">
        </form>
                </td>
        </tr>
</table>
</body>
</html>
```

15.14.5 HTML STANDARD MENU DESIGN-FILENAME: MENU0.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /index.htm
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
<meta name="author" content="Achema Egbunu">
<title>PCC60HzStandardMenu</title>
</head>

<body>

Standard menu<ul>
```

```
                <li><a href="SetPts.htm?gB2=1" target="Hauptframe">Setpoints</a>
<em>
                <br>values and <br>control modes</em></li>
                <li>Logging is off</li>
                <li><a href="Menu1.htm?gB2=1&gB9=2">Start</a> <br><em>last log
will be
                overwritten</em></li>
                <li><a href="log.dat?gB2=2" download>download CSV</a><br /><em>
                ASCII-File with unsigned<br>comma separated values
                </em>
                <li>
                <a href="log.dat" download>download raw</a><br>binary format (no
                separator):<br>long_32
                <strong>time</strong><br>int_16 <strong>R</strong><br>int_16
<strong>S</strong><br>
                int_16
                <strong>T</strong></ul>

<p> </p>
<ul>
        <li>Self</li>
        <li><a href="Menu3.htm?gB2=1">Service</a></li>
</ul>

</body>

</html>
```

15.14.6 HTML LOGGING DESIGN-FILENAME: MENU1.HTM

HTTP/1.1 200 OK

Content-Type: text/html

Content-Location: /index.htm

Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">

<head>

<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">

<meta content="de" http-equiv="Content-Language">

<meta http-equiv="expires" content="0">

<meta name="author" content="Achema Egbunu">

<title>PCC60HzLoggingMenu</title>

</head>

<body>

Standard menu

Setpoints

values and
control modes

Stop Logging

<!-- complete -->

enables download

<img src="http://www.netanimations.net/Guestbook-04-june.gif">

```
<p>WARNING <br>
any webserver stress leads to recording gaps</p>

</body>

</html>
```

15.14.7 HTML SERVICE MENU DESIGN-FILENAME: MENU3.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /index.htm
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
<meta name="author" content="Achema Egbunu">

<title>PCC60HzServiceMenu</title>
</head>

<body>

<p>Service menu<br />
for experts only! <a href="Menu0.htm">Ok, Bye</a></p>
```

```
<p> </p>
<p> </p>
<ul>
                    <li><a  href="Arrays.htm?gB2=1"  target="Hauptframe">System
status</a><br/>
                    <em>(shared arrays)</em></li>
                    <li><a                                  href="PID.htm?gB2=1"
target="Hauptframe">(P),(I),(D)</a>
                    <em><br />k faktors </em></li>
                    <li><a                                href="HWadj150.htm?gB2=1"
target="Hauptframe">Hardware Offset</a><br />
                    <em>Displays sampled channel values and reference value within
linear
                    range of ±150VDC</em></li>
          </ul>

</body>

</html>
```

15.14.8 HTML SYSTEM STATUS-FILENAME: ARRAYS.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /index.htm
Connection: close

<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
```

```
<meta name="author" content="Achema Egbunu">
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<title>PCC60Hz SystemStatus</title>
</head>

<body>
<p style="background-color:red;color:yellow">Warning, manipulations on this page are taken without any control.
Wrong manipulations can lead to program crashes or even damage
to hardware.</p>
<a href="Arrays.htm?gB2=1">refresh values</a> or <a href="Arrays.htm?gB2=0">
getWildcards</a><h3>Global Byte Array </h3>
<table>
        <tr style="background-color:yellow;">
                <td>Array<br>index</td>
                <td style="width: 130px">Meaning of Index</td>
                <td style="width: 64px">place<br>holder</td>
                <td style="width: 508px">Meaning of content of volatile byte
                gArByte[11];</td>
        </tr>
        <tr>
                <td>0</td>
                <td style="width: 130px">logSemaphore 0</td>
                <td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gB0" type="text" value="#gB0#" style="height: 22px">
        <input type="hidden" name="gB2" value="1"> </form>
                </td>
```

```
            <td style="width: 508px">LogOff 0; WriteProgress 1; GetFirst 2; GetNext
            3</td>
        </tr>
        <tr style="background-color:silver">
            <td>1</td>
            <td style="width: 130px">ADCRun 1</td>
            <td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gB1" type="text" value="#gB1#">
        <input type="hidden" name="gB2" value="1"></form>
            </td>
            <td style="width: 508px">Channel just scanned</td>
        </tr>
        <tr>
            <td>2</td>
            <td style="width: 130px"><strong>WebAnswerMode </strong></td>
            <td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gB2" type="text" value="#gB2#">
        <input type="hidden" name="gB2" value="1"></form>
            </td>
            <td style="width: 508px"><strong>AMRaw 0; AMParsedHTML 1; AMCSV 2
            </strong> </td>
        </tr>
        <tr>
            <td style="height: 23px">3</td>
```

```
<td style="width: 130px; height: 23px;">IdxSetDuty </td>
<td style="width: 64px; height: 23px;">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB3" type="text" value="#gB3#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px; height: 23px;">Setpoint for open loop control</td>
</tr>
<tr style="background-color:silver">
<td>4</td>
<td style="width: 130px">IdxControlMode</td>
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB4" type="text" value="#gB4#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px">OpenLoopControl 0; PidControl 1; FuzzyControl
2; NN 3; Test 4 </td>
</tr>
<tr>
<td>5</td>
<td style="width: 130px">PIDKP</td>
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB5" type="text" value="#gB5#">
```

```
<input type="hidden" name="gB2" value="1"></form>
        </td>
        <td style="width: 508px">parameter for PID-Control</td>
</tr>
<tr style="background-color:silver">
        <td>6</td>
        <td style="width: 130px">PIDKI</td>
        <td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB6" type="text" value="#gB6#">
<input type="hidden" name="gB2" value="1"></form>
        </td>
        <td style="width: 508px">parameter for PID-Control</td>
</tr>
<tr>
        <td>7</td>
        <td style="width: 130px">PIDKD</td>
        <td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB7" type="text" value="#gB7#">
<input type="hidden" name="gB2" value="1"></form>
        </td>
        <td style="width: 508px">parameter for PID-Control</td>
</tr>
<tr style="background-color:silver">
        <td>8</td>
        <td style="width: 130px">PWMDuty</td>
```

```
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB8" type="text" value="#gB8#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px">backuped actual duty cycle (at PwmOutPin 3) </td>
</tr>
<tr>
<td>9</td>
<td style="width: 130px"><strong>IdxCmd</strong></td>
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB9" type="text" value="#gB9#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px"><strong>CmdNone 0; CmdSetControl 1; CmdStartLog
2; CmdStopLog 3
<br>(set to 0 after execution)</strong></td>
</tr>
<tr style="background-color:silver">
<td>10</td>
<td style="width: 130px">ControlTrigger<br>actual value</td>
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB10" type="text" value="#gB10#">
```

```
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px">count down for change of Duty cycle in control</td>
</tr>
<tr>
<td>11</td>
<td style="width: 130px">ControlTrigger<br>interval</td>
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB11" type="text" value="#gB11#" style="height: 22px">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px">sequence of Duty cycle in control (start value
of count down)</td>
</tr>
<tr style="background-color:silver">
<td>12</td>
<td style="width: 130px">byte spy</td>
<td style="width: 64px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB12" type="text" value="#gB12#" style="height: 22px">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 508px">option to inspect or change any by value by
inserting a simple command <br>in the source code for debug purpose.</td>
</tr>
```

```
</table>

<h3>Global Integer Array</h3>
<table>
	<tr style="background-color:lime">
		<td style="height: 23px; width: 55px">Array<br>index</td>
		<td style="height: 23px; width: 63px">Meaning of Index</td>
		<td style="height: 23px; width: 51px">place<br>holder</td>
		<td style="height: 23px; width: 328px;">Meaning of content of volatile
		int gArInt[ngINT];</td>
	</tr>
	<tr>
		<td style="width: 55px; height: 23px;">0</td>
		<td style="width: 63px; height: 23px;"> R </td>
		<td style="width: 51px; height: 23px;">
<form action="Arrays.htm" method="get" style="width: 150px; " title="Arduino 60Hz Control">
	<input name="gI0" type="text" value="#gI0#"/>
	<input type="hidden" name="gB2" value="1"> </form>
		</td>
		<td style="height: 23px; width: 328px;">sampled Value for phase R directly</td>
	</tr>
	<tr style="background-color:silver">
		<td style="width: 55px">1</td>
		<td style="width: 63px">S </td>
		<td style="width: 51px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
```

```
<input name="gI1" type="text" value="#gI1#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 328px">sampled Value for phase S directly</td>
</tr>
<tr>
<td style="width: 55px">2</td>
<td style="width: 63px">T </td>
<td style="width: 51px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gI2" type="text" value="#gI2#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="width: 328px">sampled Value for phase T directly</td>
</tr>
<tr style="background-color:silver">
<td style="height: 23px; width: 55px">3</td>
<td style="height: 23px; width: 63px">BufR</td>
<td style="height: 23px; width: 51px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gI3" type="text" value="#gI3#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="height: 23px; width: 328px;">buffered to have it consistent</td>
</tr>
<tr>
```

```
                <td style="width: 55px">4</td>
                <td style="width: 63px">BufS</td>
                <td style="width: 51px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gI4" type="text" value="#gI4#" />
        <input type="hidden" name="gB2" value="1"></form>
                </td>
                <td style="width: 328px">buffered to have it consistent</td>
        </tr>
        <tr style="background-color:silver">
                <td style="width: 55px">5</td>
                <td style="width: 63px">BufT</td>
                <td style="width: 51px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gI5" type="text" value="#gI5#">
        <input type="hidden" name="gB2" value="1"></form>
                </td>
                <td style="width: 328px">buffered to have it consistent</td>
        </tr>
</table>
<h3>Global Long Integer Array</h3>
<table>
        <tr style="background-color:aqua">
                <td style="width: 97px; height: 42px;">Array<br>index</td>
                <td style="width: 167px; height: 42px;">Meaning<br>of Index</td>
                <td style="width: 204px; height: 42px;">place<br>holder</td>
                <td style="height: 42px">Meaning of content</td>
```

```
</tr>
<tr>
    <td style="width: 97px">0</td>
    <td style="width: 167px">Measure </td>
    <td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gL0" type="text" value="#gL0#">
<input type="hidden" name="gB2" value="1"></form>
    </td>
    <td>newest or previous (toggeled)</td>
</tr>
<tr style="background-color:silver">
    <td style="width: 97px">1</td>
    <td style="width: 167px">Measure</td>
    <td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gL1" type="text" value="#gL1#">
<input type="hidden" name="gB2" value="1"></form>
    </td>
    <td>newest or previous (toggeled)</td>
</tr>
<tr>
    <td style="height: 23px; width: 97px">2</td>
    <td style="height: 23px; width: 167px">Filter </td>
    <td style="height: 23px; width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
```

```
<input name="gL2" type="text" value="#gL2#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td style="height: 23px">newest or previous (toggeled)</td>
</tr>
<tr style="background-color:silver">
<td style="width: 97px">3</td>
<td style="width: 167px">Filter </td>
<td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gL3" type="text" value="#gL3#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td>newest or previous (toggeled)</td>
</tr>
<tr>
<td style="width: 97px">4</td>
<td style="width: 167px">DMeasure</td>
<td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
<input name="gL4" type="text" value="#gL4#">
<input type="hidden" name="gB2" value="1"></form>
</td>
<td>derivative</td>
</tr>
<tr style="background-color:silver">
<td style="height: 23px; width: 97px;">5</td>
```

```
                <td style="height: 23px; width: 167px;">DFilter </td>
                <td style="height: 23px; width: 204px;">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL5" type="text" value="#gL5#">
        <input type="hidden" name="gB2" value="1"></form>
                </td>
                <td style="height: 23px">derivative</td>
        </tr>
        <tr>
                <td style="width: 97px">6</td>
                <td style="width: 167px">IdxSetPM </td>
                <td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL6" type="text" value="#gL6#">
        <input type="hidden" name="gB2" value="1"></form>
                </td>
                <td>Setpoint as unit measure</td>
        </tr>
        <tr style="background-color:silver">
                <td style="height: 21px; width: 97px;">7</td>
                <td style="height: 21px; width: 167px;">DCOffset0 </td>
                <td style="height: 21px; width: 204px;">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL7" type="text" value="#gL7#">
        <input type="hidden" name="gB2" value="1"></form>
                </td>
```

```
            <td style="height: 21px">computed Value for analog input as 0V</td>
        </tr>
        <tr>
            <td style="width: 97px">8</td>
            <td style="width: 167px">Idle </td>
            <td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL8" type="text" value="#gL8#">
        <input type="hidden" name="gB2" value="1"></form>
            </td>
            <td>number of idle loops in main </td>
        </tr>
        <tr style="background-color:silver">
            <td style="width: 97px">9</td>
            <td style="width: 167px"> LogTime</td>
            <td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL9" type="text" value="#gL9#">
        <input type="hidden" name="gB2" value="1"></form>
            </td>
            <td>time axis for logged values (gaps might happen)</td>
        </tr>
        <tr>
            <td style="width: 97px">10</td>
            <td style="width: 167px">LogTimeBuf</td>
            <td style="width: 204px">
```

```
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL10" type="text" value="#gL10#">
        <input type="hidden" name="gB2" value="1"></form>
                </td>
                <td>consistent ot samples</td>
        </tr>
        <tr style="background-color:silver">
                <td style="width: 97px">11</td>
                <td style="width: 167px">SetMeasure </td>
                <td style="width: 204px">
<form action="Arrays.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
        <input name="gL11" type="text" value="#gL11#">
        <input type="hidden" name="gB2" value="1"></form>
                </td>
                <td>setpoint given as unit measure</td>
        </tr>
</table>


</body>


</html>
```

15.14.9 HTML CONTROL- FILENAME: PID.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /index.htm
Connection: close
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta content="de" http-equiv="Content-Language">
<meta http-equiv="expires" content="0">
<meta name="author" content="Achema Egbunu">

<title>PCC60HzControlMenuPID</title>
</head>

<body>

<h3>PID-Control</h3>
<table>
	<tr>
		<td style="width: 17px">
		 KP</td>
		<td>
		 KI</td>
		<td>
		 KD</td>
	</tr>
	<tr>
		<td style="width: 17px">
<form action="PID.htm" method="get" style="width: 50px" title="SetPoints">
	<input name="gB5" type="text" value="#gB5#" style="width: 50px">
	<input type="hidden" name="gB2" value="1">
```

```
<input type="hidden" name="gB4" value="1"></form>
</td>
<td>
<form action="PID.htm" method="get" style="width: 50px" title="Arduino 60Hz Control">
<input name="gB6" type="text" value="#gB6#" style="width: 50px">
<input type="hidden" name="gB2" value="1">
<input type="hidden" name="gB4" value="1"></form>
</td>
<td>
<form action="PID.htm" method="get" style="width: 50px; height: 22px;" title="Arduino 60Hz Control">
<input name="gB7" type="text" value="#gB7#" style="width: 50px">
<input type="hidden" name="gB2" value="1">
<input type="hidden" name="gB4" value="1"></form>
</td>
</tr>
</table>
Enter <strong>integer</strong> values within range: 0...255<p> </p>
 <table border="2">
<tr>
<td style="height: 23px">setpoint duty cycle (0:255)<br/>for open loop
control</td>
<td style="height: 23px">setpoint measure<br/>for PID-Control</td>
</tr>
<tr>
<td>
<form action="PID.htm" method="get" style="width: 150px; height: 22px;" title="Arduino 60Hz Control">
```

```
<input name="gB4" type="text" value="#gB4#">
<input type="hidden" name="gB2" value="1">
<input type="hidden" name="gB4" value="0"></form>
</td>
<td>
<form action="PID.htm" method="get" style="width: 150px" title="Arduino 60Hz Control">
<input name="gL11" type="text" value="#gL11#">
<input type="hidden" name="gB2" value="1">
<input type="hidden" name="gB4" value="1"></form>
</td>
</tr>
</table>

</body>

</html>
```

15.14.10 HTML HARDWARE OFFSET-FILENAME: HWADJ150.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /adjPt150.htm
Connection: close

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
<head>
<title>PCC60HzCalibrateHW150V</title>
<meta name="author" content="Achema Egbunu">
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta http-equiv="expires" content="0">
<meta http-equiv="refresh" content="1">
</head>



<body>
<table border="2">
	<tr>
		<td style="width: 150px;font-size:xx-large">actual</td>
		<td style="width: 150px;font-size:xx-large">#gI0#</td>
		<td style="width: 150px;font-size:xx-large">#gI1#</td>
		<td style="width: 150px;font-size:xx-large">#gI2#</td>
	</tr>
	</table>
</body>

</html>
```

15.14.11 HTML HARDWARE OFFSET ACTUAL-FILENAME: FRRST0.HTM

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Location: /adjPt150.htm
Connection: close
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<head>
<title>PCC60HzCalibrateHW150V</title>
<meta name="author" content="Achema Egbunu">
<meta content="text/html; charset=windows-1252" http-equiv="Content-Type">
<meta http-equiv="expires" content="0">
<meta http-equiv="refresh" content="1">
</head>


<body>
<table border="2">
	<tr>
		<td style="width: 150px;font-size:xx-large">actual</td>
		<td style="width: 150px;font-size:xx-large">#gI0#</td>
		<td style="width: 150px;font-size:xx-large">#gI1#</td>
		<td style="width: 150px;font-size:xx-large">#gI2#</td>
	</tr>
	</table>
</body>

</html>
```

15.15 APPENDIX 15: ADJUSTMENT OF THE CONTROLLER COEFFICIENTS

15.15.1 P CONTROLLER COEFFICIENT: KP

```
% ########################################################
% Adjustment of Control Parameters (KP, KI & KD)   #######
% ACHEMA HOSEA EGBUNU (142773)                     #######
% ########################################################
close all; clear all; clc; %clear the command window, work space & history
%setpoint = 120;
```

```
           %File index number which is part of the filename
%-------------------------------------------------------------------------%
Kp = [1, 64, 132, 136, 144, 160, 192, 255];
Prefix='S23KP';PostFix='.TXT';  %Files from from the measurement

n = length(Kp);
```

```
N = 9;               %Level of wavelet decomposition, increase to smoothing
wName='db5';         %Specific wavelet name used

%Prefix='OpenLoop';PostFix='.TXT';  %Files from from the measurement

for k = 1:n
```

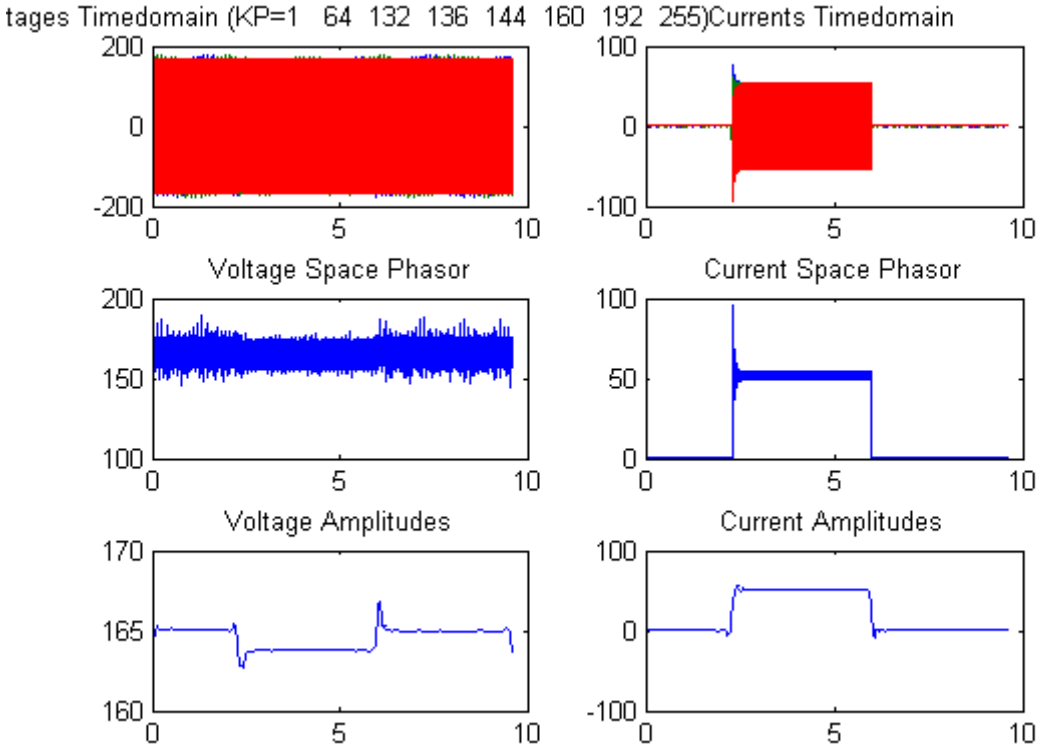
```
      KP = Kp(k);
      Raw = load([Prefix,num2str(KP),'.txt']);  %Load the file
      t = Raw(:,1);        % First column is the time axis
      RST = Raw(:,2:4);    % R, S, T Phases
      I_rst = Raw(:,5:7);  % Current for each three phases(I_r, I_s, I_t)

    % Computing the SPACE PHASOR to find a measure for Amplitude
    % and a hard wavelet filter to remove the noise
    % Call function that compute space phsor for voltage and current
    Usp=abs(ComputeSpacePhasor(RST));   %Voltage Space Phaseor
    Isp=abs(ComputeSpacePhasor(I_rst)); %Current Space Phasor
```

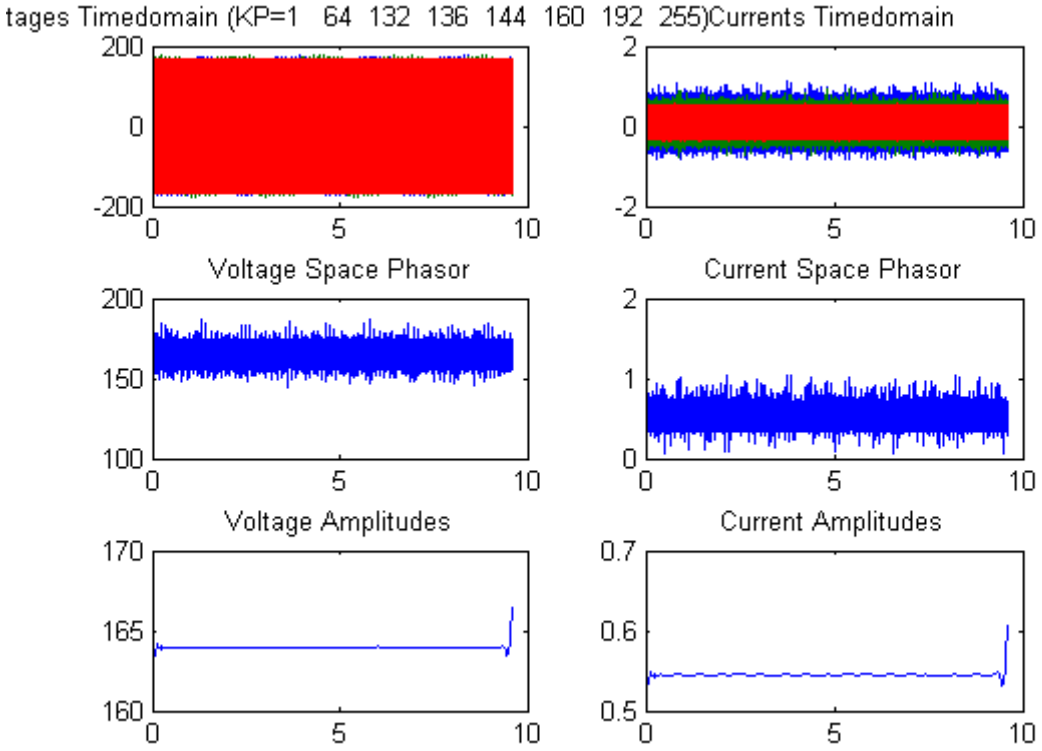
```
    %Filtering the Noise from the measurements
    [C,L] = wavedec(Usp,N,wName); %Perform wavelet decomposition of Usp
    C(L(1)+1:end)=0;
    FiltUsp = waverec(C,L,wName); %Perform wavelet reconstruction of Usp
    [C,L] = wavedec(Isp,N,wName); %Perform wavelet decomposition of Isp
    C(L(1)+1:end)=0;
    FiltIsp = waverec(C,L,wName); %Perform wavelet reconstruction of Isp
```

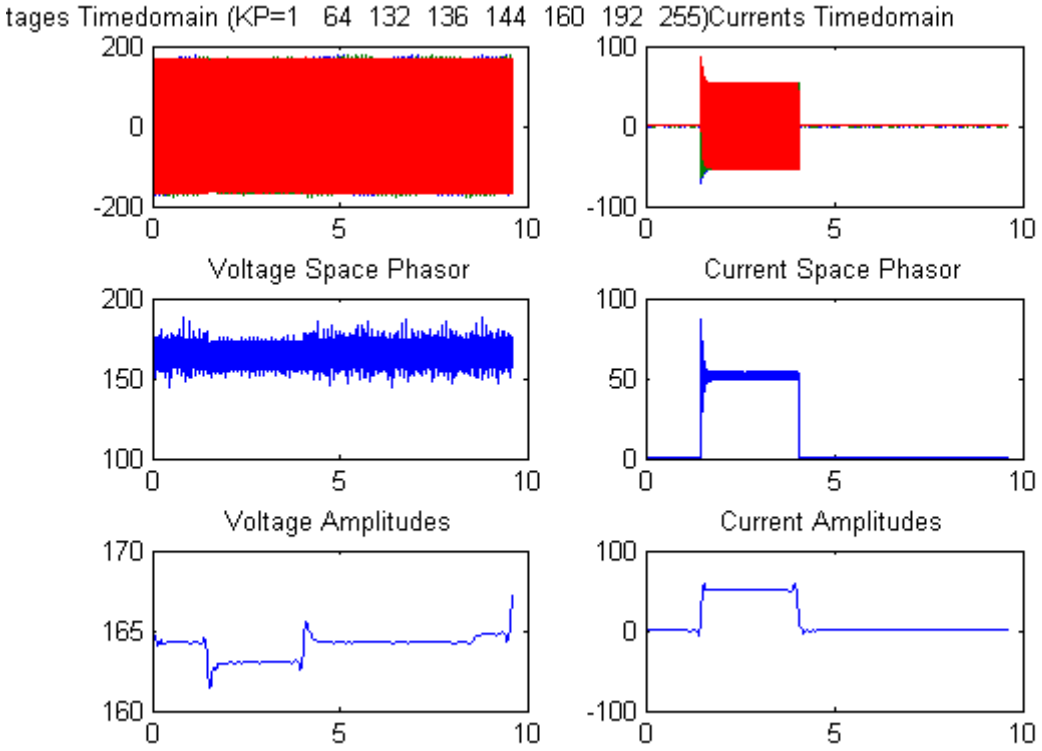
Show Corresponding Plots

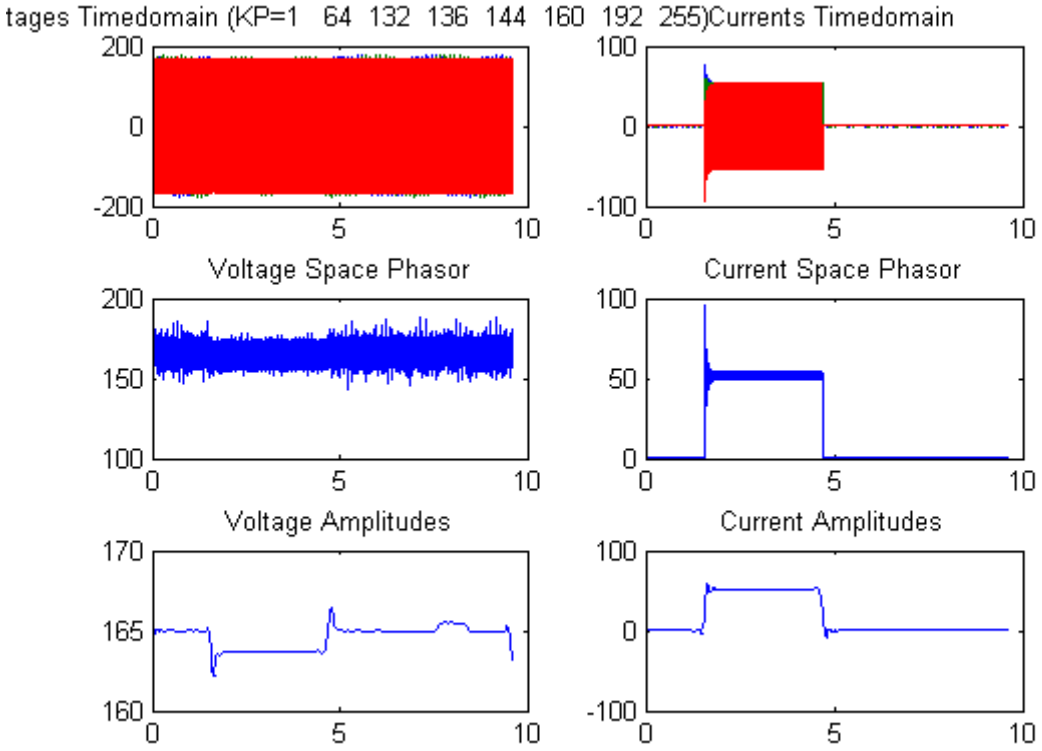
```
    h=figure (k);
    subplot(3,2,1); plot(t,RST);
    title(['Voltages Timedomain (KP=',num2str(Kp),')']);
```

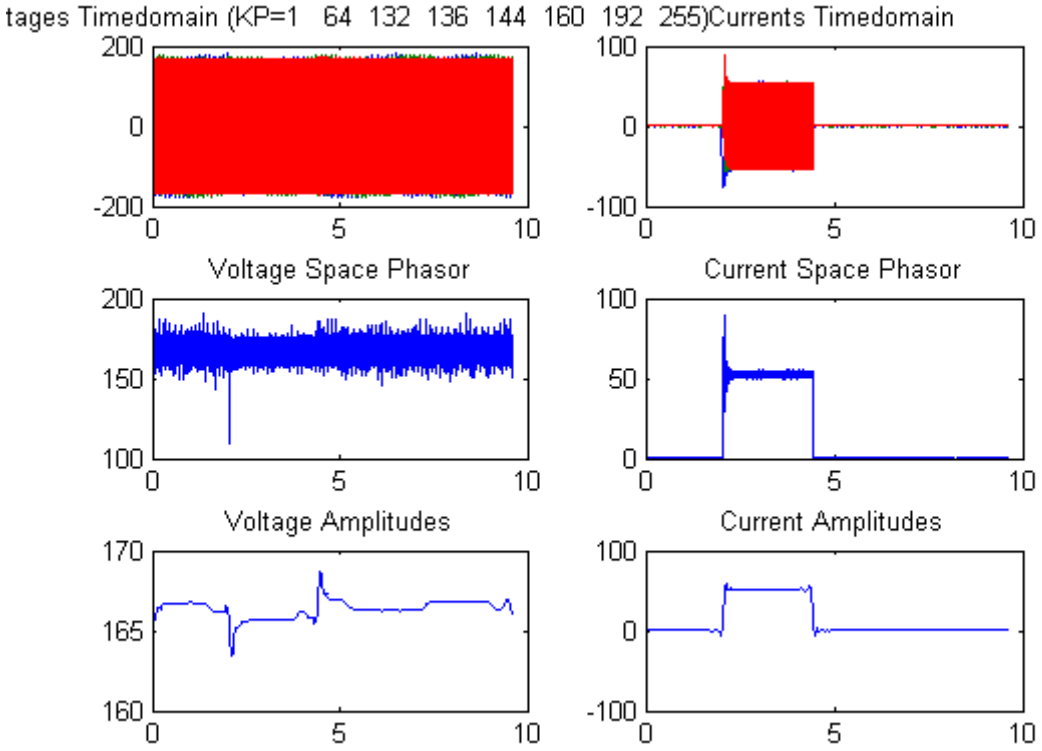
```
    subplot(3,2,2); plot(t,I_rst); title('Currents Timedomain');
    subplot(3,2,3); plot(t,Usp); title('Voltage Space Phasor');
    subplot(3,2,4); plot(t,Isp); title('Current Space Phasor');
    subplot(3,2,5); plot(t,FiltUsp); title('Voltage Amplitudes');
    subplot(3,2,6); plot(t,FiltIsp); title('Current Amplitudes');
```

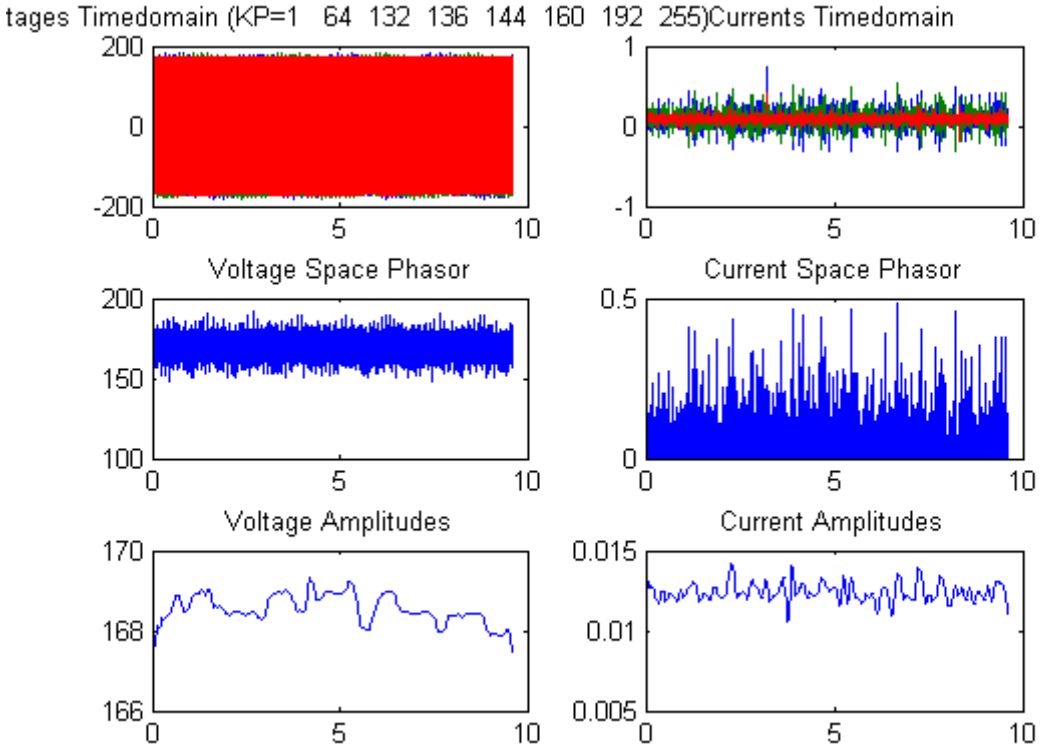


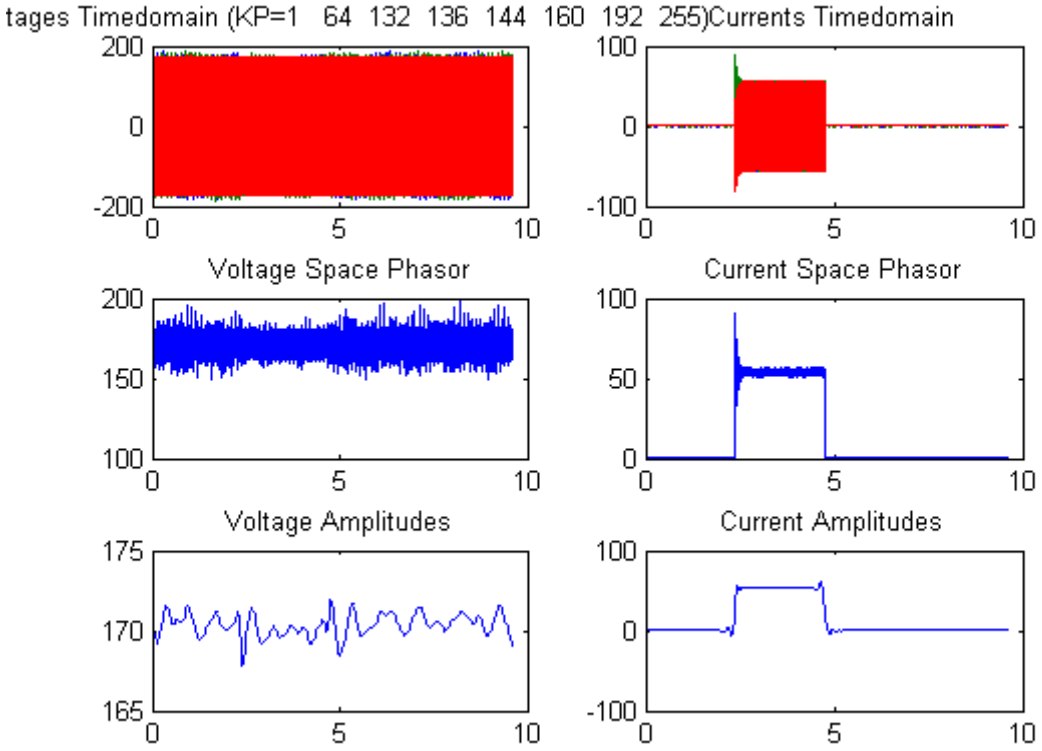

tages Timedomain (KP=1 64 132 136 144 160 192 255)Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes

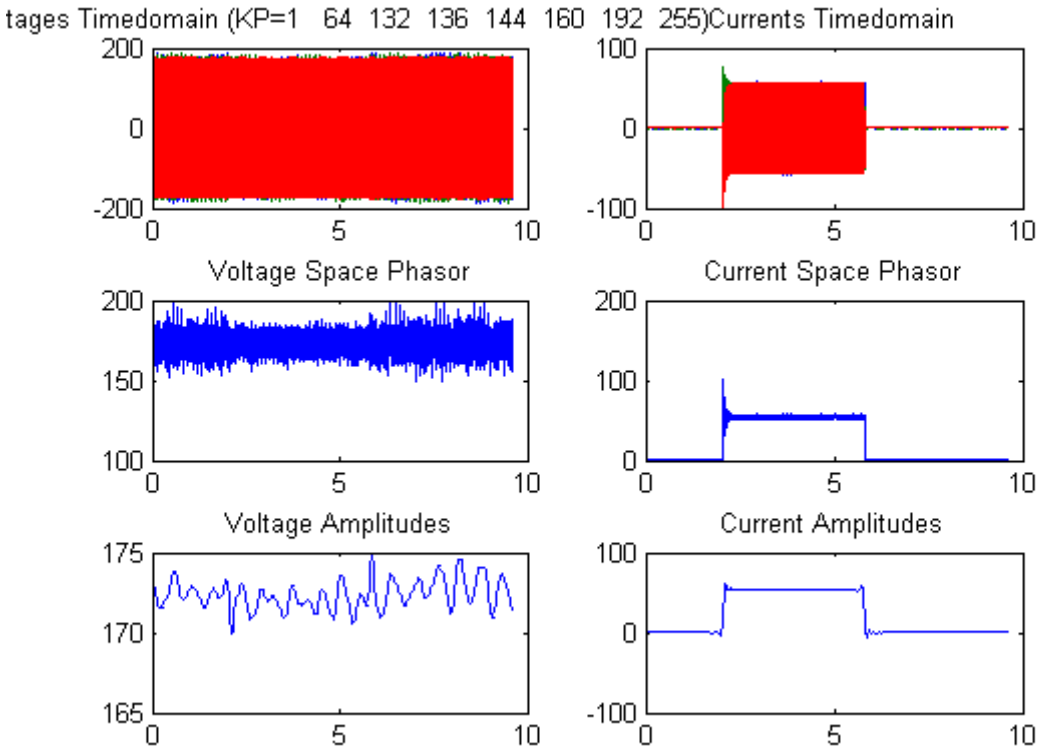

tages Timedomain (KP=1 64 132 136 144 160 192 255)Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes


tages Timedomain (KP=1 64 132 136 144 160 192 255)Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes


tages Timedomain (KP=1 64 132 136 144 160 192 255)Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes


tages Timedomain (KP=1 64 132 136 144 160 192 255)Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes


tages Timedomain (KP=1 64 132 136 144 160 192 255)Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes




```
end
```

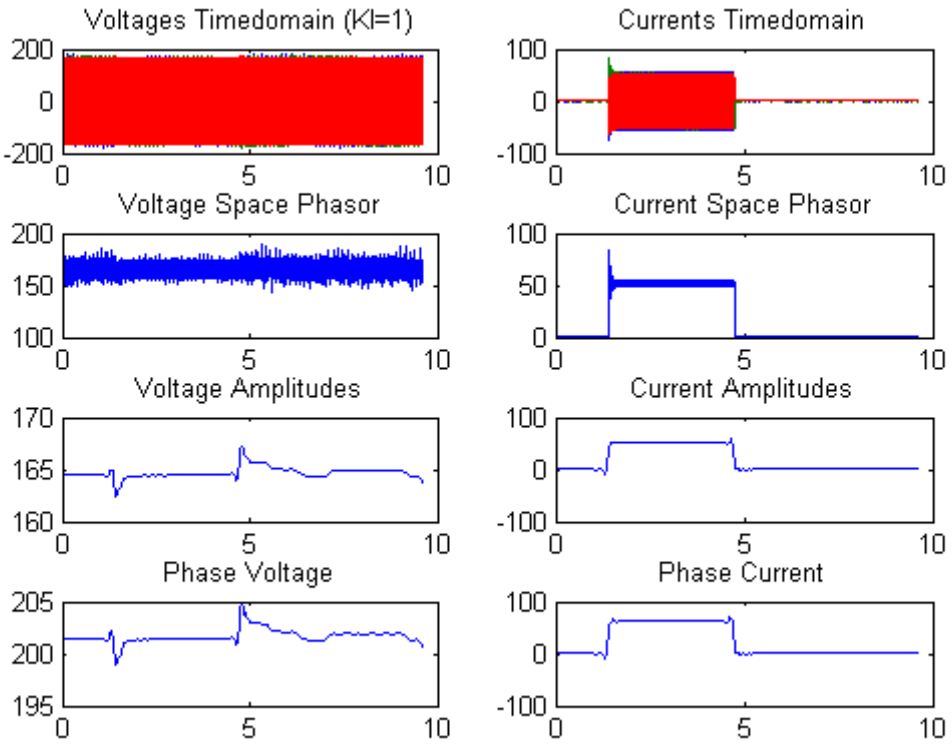
Published with MATLAB® R2013a

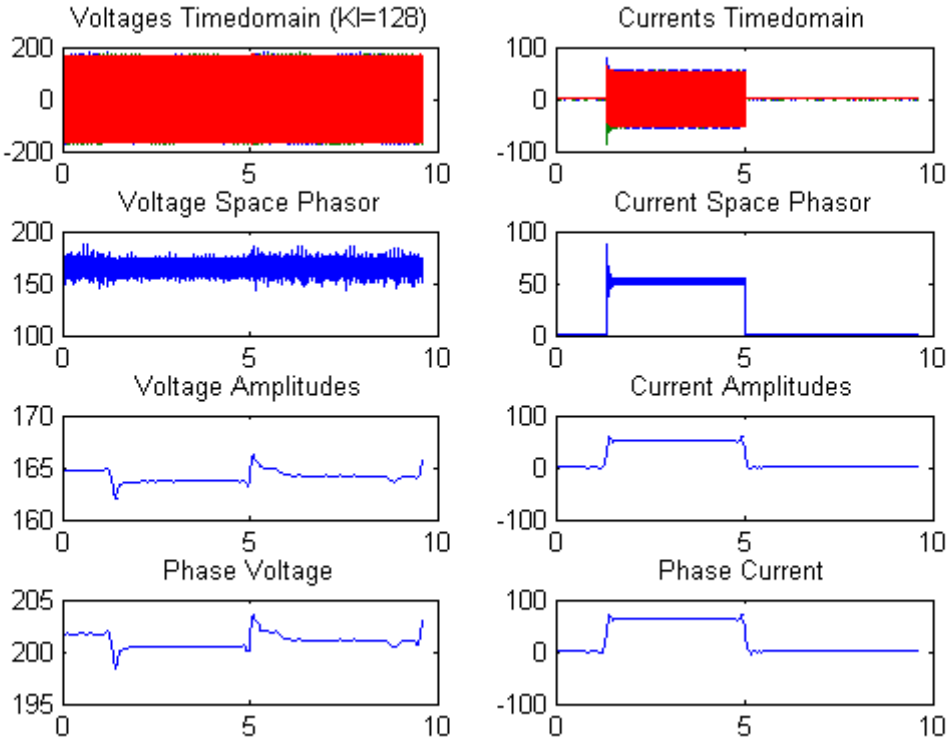
```
% ######################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)             #######
% #######            SYSTEMS AND CONTROL ENGINEERING            #######
% #######          FUNCTION FOR COMPUTING SPACE PHASOR          #######
% ######################################################################

%%
     %Function For Computing the Space Phasor
%-------------------------------------------------%
function SP = ComputeSpacePhasor(samples)
SP = 2/3*(samples(:,1) + samples(:,2)*exp(1i*2*pi/3) +
samples(:,3)*exp(1i*4*pi/3));
```

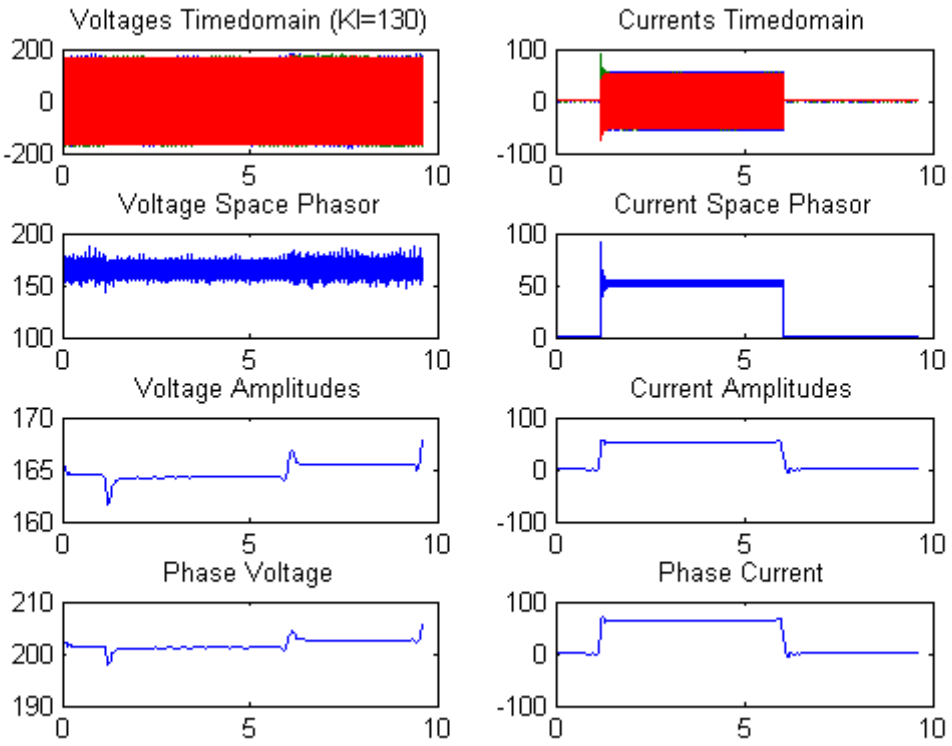
15.15.2 PI CONTROLLER COEFFICIENT: KI

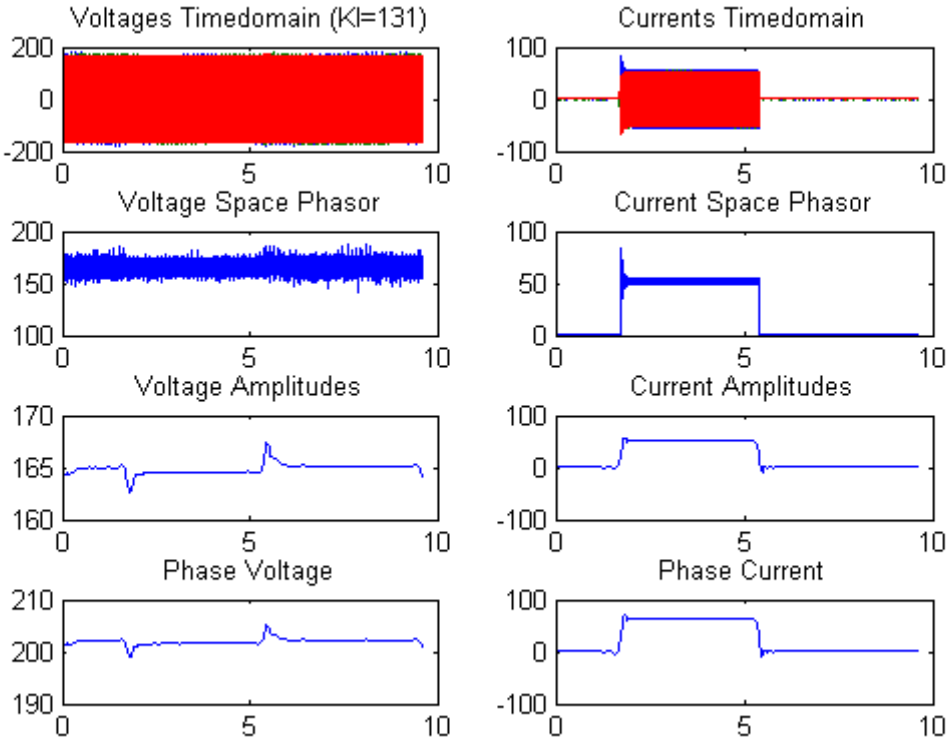
```
% ######################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)             #######
% #######            SYSTEMS AND CONTROL ENGINEERING            #######
% #######            ANALYSIS OF THE MEASUREMENT DATA           #######
% #######    Adjustment of Control Parameter (KI)#######
% ######################################################################
```

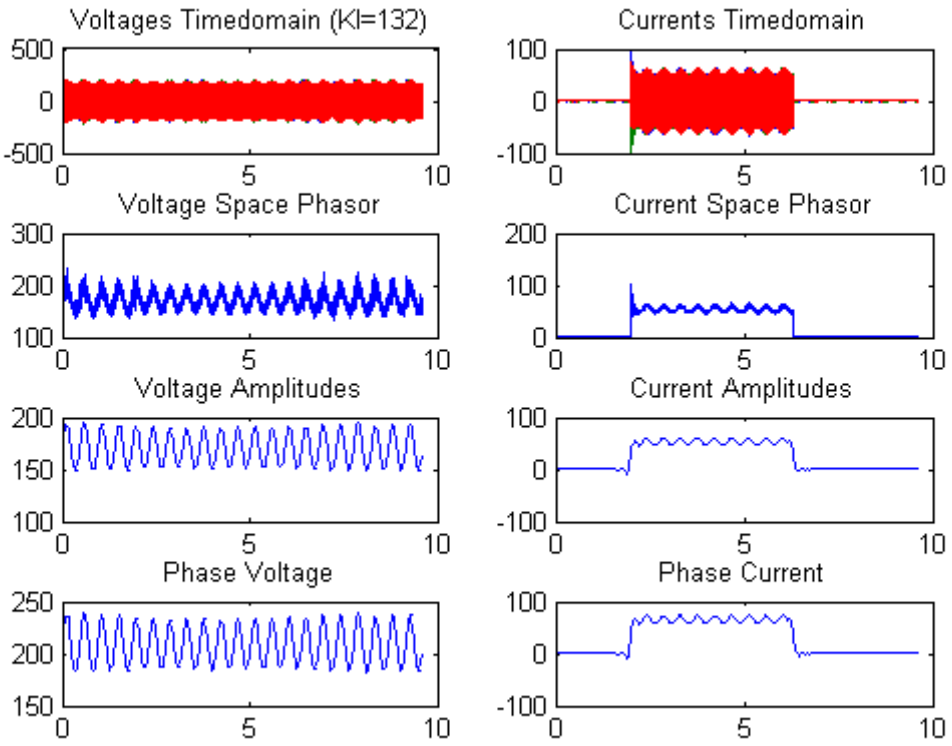
```
close all; clear all; clc; %clear the command window, work space & history
%setpoint = 120;

          %File index number which is part of the filename
%-------------------------------------------------------------------------%
Kp = [1, 128, 130, 131, 132, 136, 144, 160, 192, 255];
Prefix='PIS22KI';PostFix='.TXT';  %Files from from the measurement

n = length(Kp);

N = 9;               %Level of wavelet decomposition, increase to smoothing
wName='db5';         %Specific wavelet name used

%Prefix='OpenLoop';PostFix='.TXT';  %Files from from the measurement

for k = 1:n
      Idx = Kp(k);
      Raw = load([Prefix,num2str(Idx),'.txt']);  %Load the file
      t = Raw(:,1);        % First column is the time axis
      RST = Raw(:,2:4);    % R, S, T Phases
      I_rst = Raw(:,5:7);  % Current for each three phases(I_r, I_s, I_t)

    % Computing the SPACE PHASOR to find a measure for Amplitude
    % and a hard wavelet filter to remove the noise
    % Call function that compute space phsor for voltage and current
    Usp=abs(ComputeSpacePhasor(RST));   %Voltage Space Phaseor
    Isp=abs(ComputeSpacePhasor(I_rst)); %Current Space Phasor

    %Filtering the Noise from the measurements
    [C,L] = wavedec(Usp,N,wName); %Perform wavelet decomposition of Usp
    C(L(1)+1:end)=0;
    FiltUsp = waverec(C,L,wName); %Perform wavelet reconstruction of Usp
    [C,L] = wavedec(Isp,N,wName); %Perform wavelet decomposition of Isp
    C(L(1)+1:end)=0;
    FiltIsp = waverec(C,L,wName); %Perform wavelet reconstruction of Isp

          %Computing the Phase to Phase Voltage and Current
    %-----------------------------------------------------------------%
    Upp = FiltUsp * sqrt(3)/sqrt(2);
    Ipp = FiltIsp * sqrt(3)/sqrt(2);

    % Show the Corresponding Plots
    h=figure;
    subplot(4,2,1); plot(t,RST);
    title(['Voltages Timedomain (KI=',num2str(Idx),')']);
    subplot(4,2,2); plot(t,I_rst);      title('Currents Timedomain');
    subplot(4,2,3); plot(t,Usp);        title('Voltage Space Phasor');
    subplot(4,2,4); plot(t,Isp);        title('Current Space Phasor');
    subplot(4,2,5); plot(t,FiltUsp);    title('Voltage Amplitudes');
    subplot(4,2,6); plot(t,FiltIsp);    title('Current Amplitudes');
    subplot(4,2,7); plot(t,Upp);        title('Phase Voltage');
    subplot(4,2,8); plot(t,Ipp);        title('Phase Current');
end
```

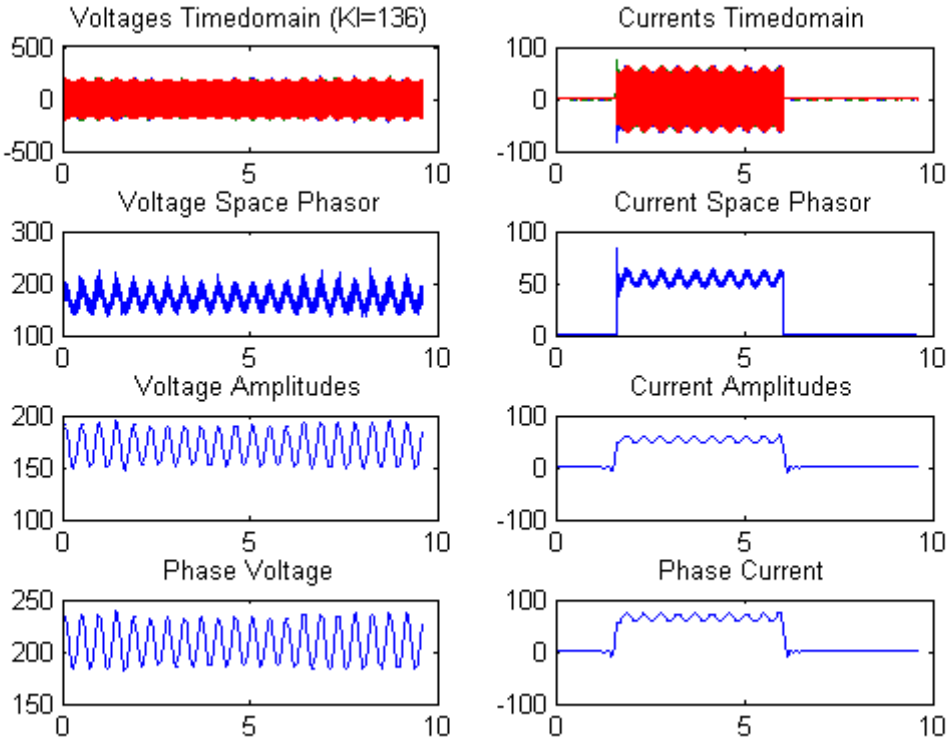

Voltages Timedomain (KI=1)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

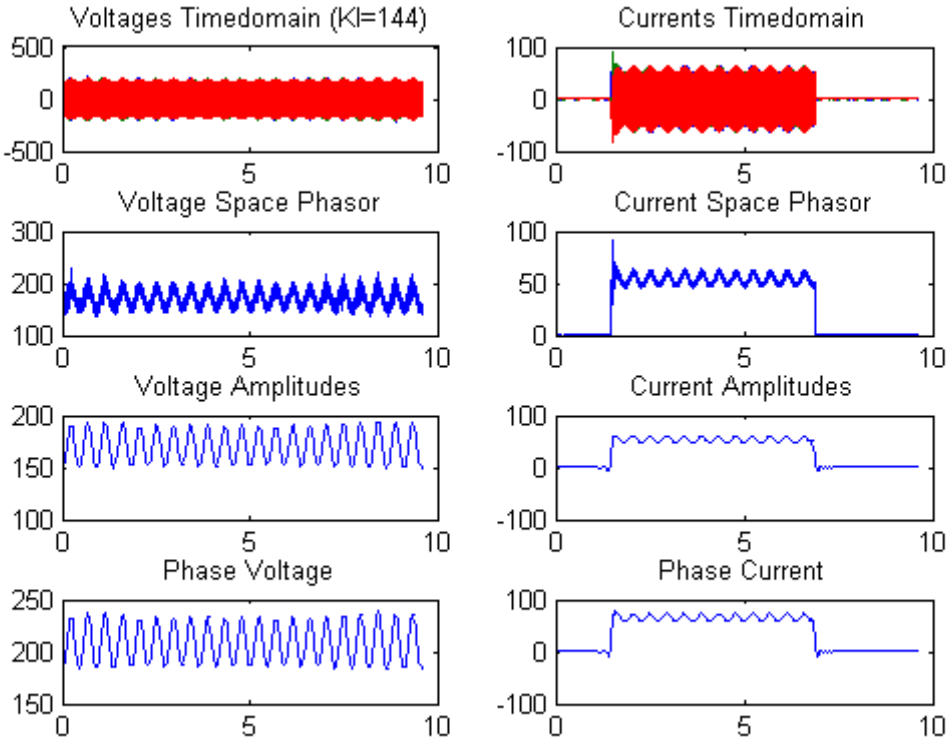

Voltages Timedomain (KI=128)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

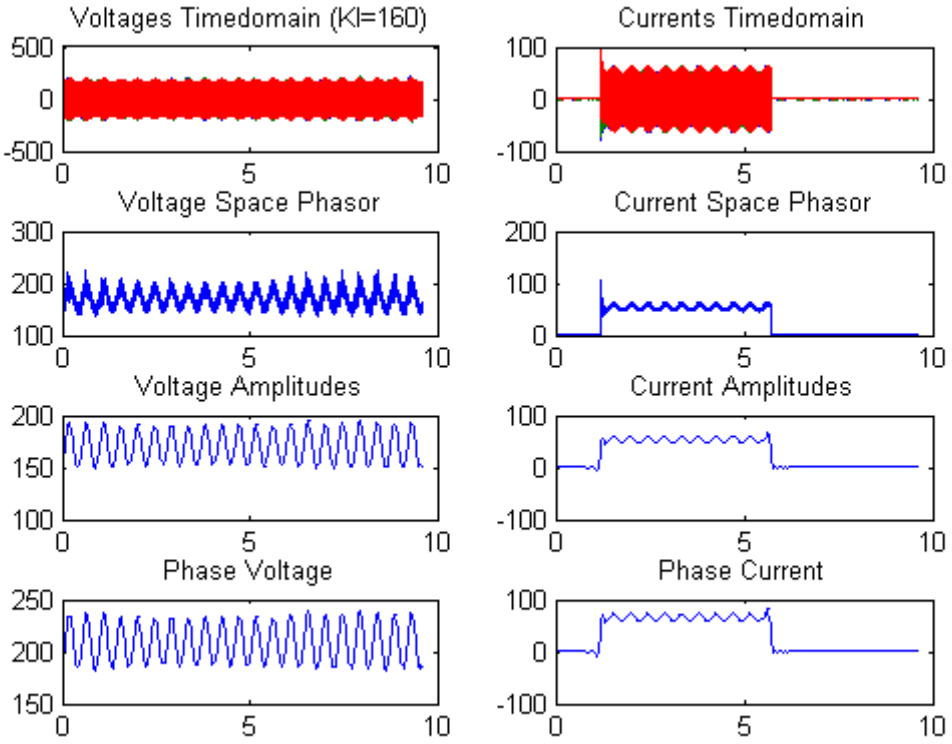

Voltages Timedomain (Kl=130)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

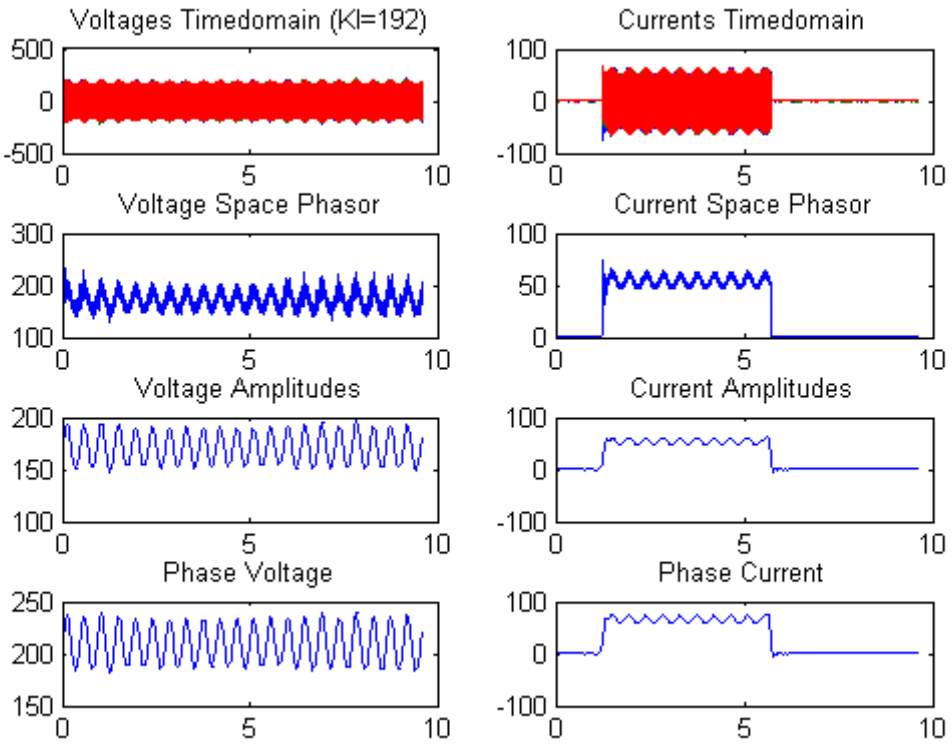

Voltages Timedomain (Kl=131)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

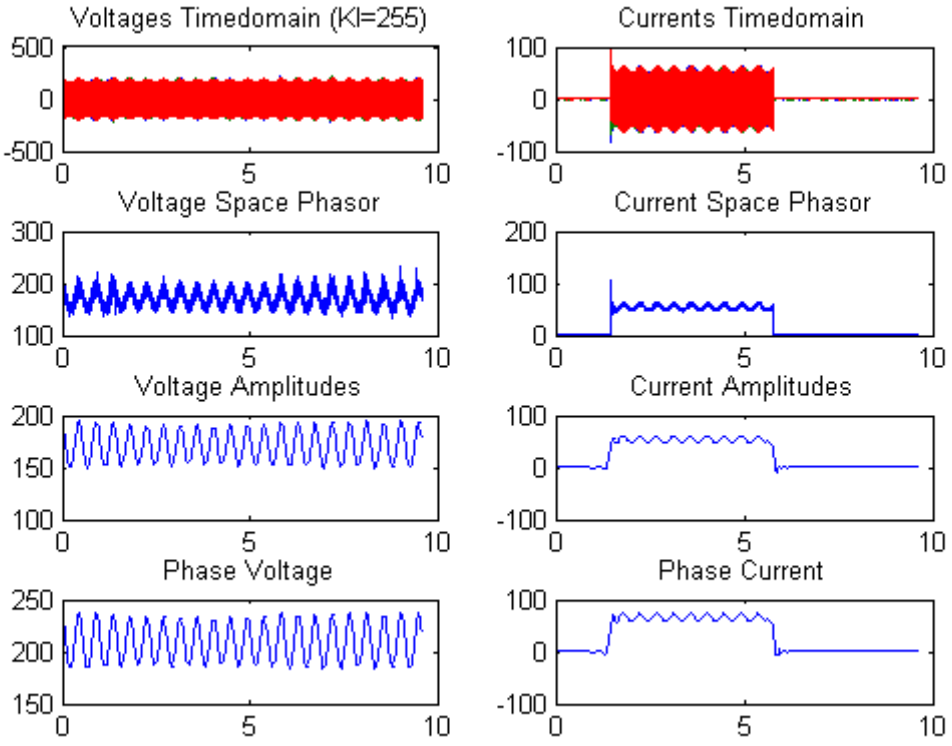

Voltages Timedomain (Kl=132)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (Kl=136)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (Kl=144)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (Kl=160)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KI=192)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KI=255)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

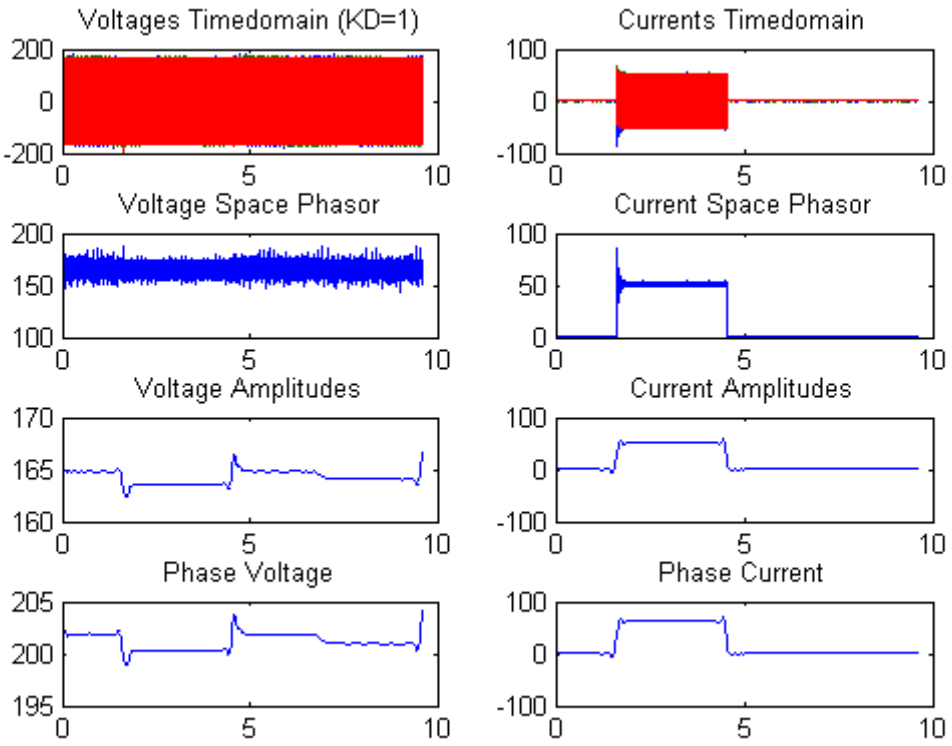
Published with MATLAB® R2013a

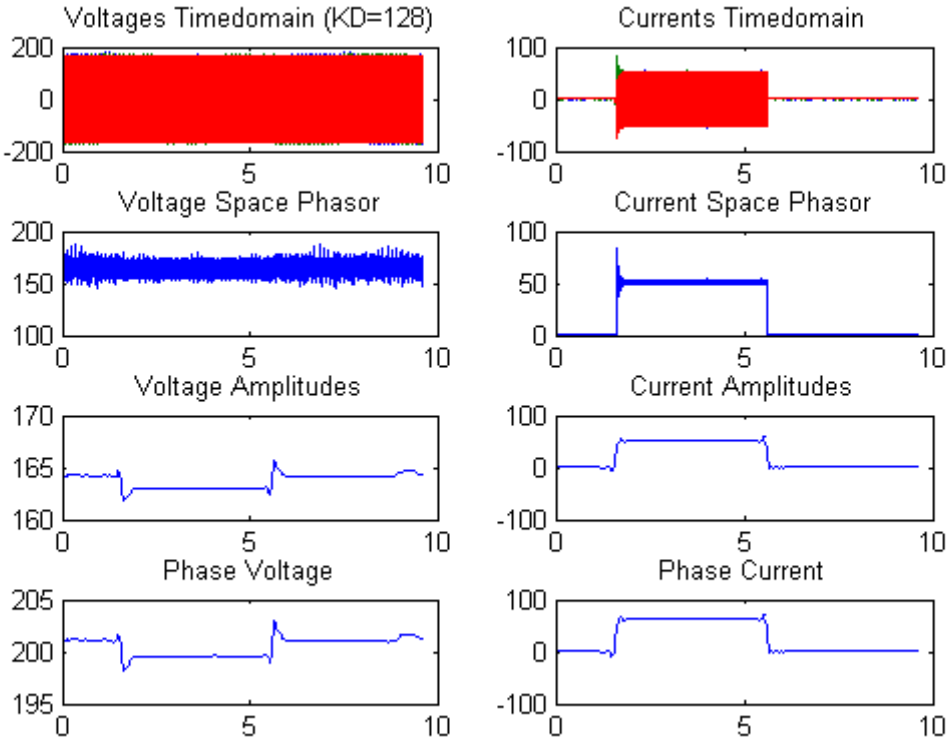
```
% ######################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)           #######
% #######            SYSTEMS AND CONTROL ENGINEERING          #######
% #######          FUNCTION FOR COMPUTING SPACE PHASOR        #######
% ######################################################################


%%
     %Function For Computing the Space Phasor
%------------------------------------------------%
function SP = ComputeSpacePhasor(samples)
SP=2/3*(samples(:,1)+samples(:,2)*exp(1i*2*pi/3)+samples(:,3)*exp(1i*4
*pi/3));
```

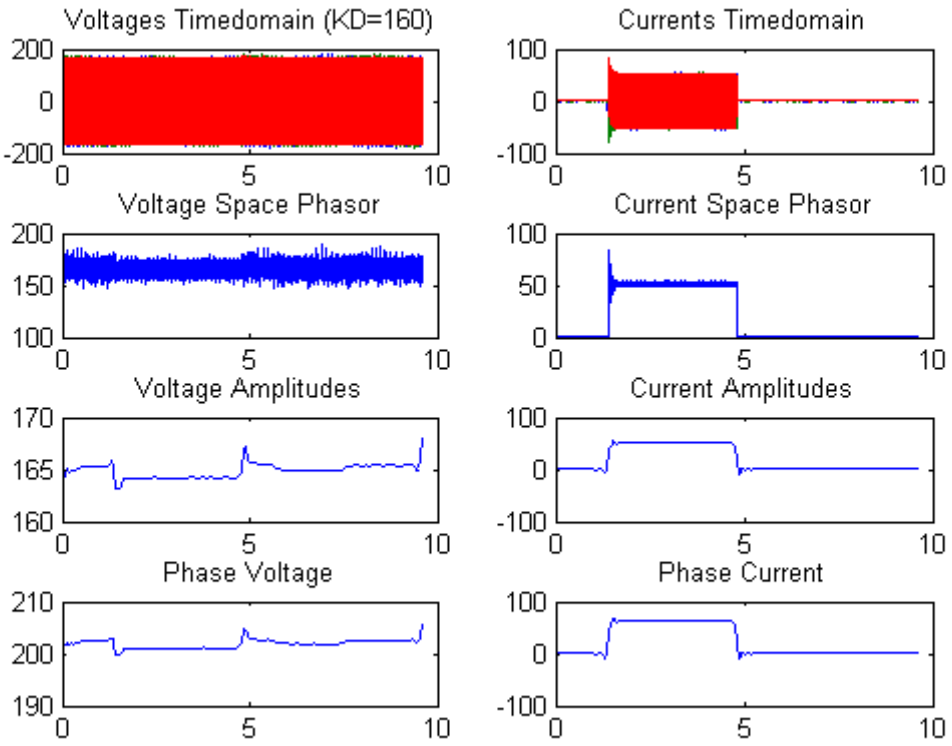
15.15.3 PID CONTROLLER COEFFICIENT: KD

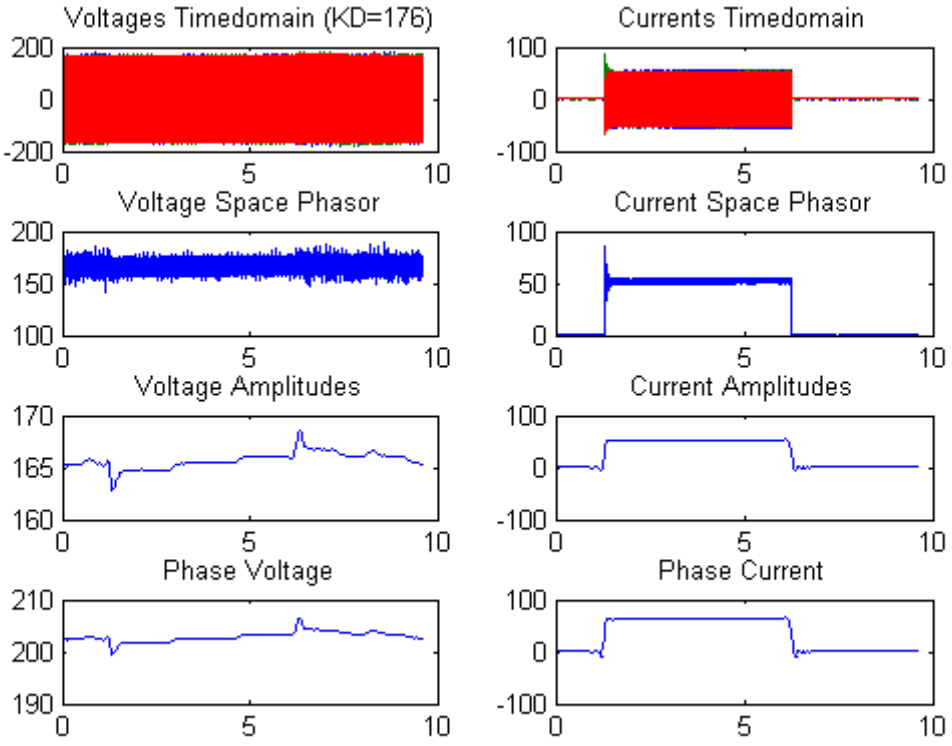
```
% ######################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)          #######
% #######            SYSTEMS AND CONTROL ENGINEERING         #######
% #######            ANALYSIS OF THE MEASUREMENT DATA        #######
% #######    Adjustment of Control Parameters (KD)          #######
% ######################################################################
close all; clear all; clc; %clear the command window, work space & history
%setpoint = 120;

          %File index number which is part of the filename
%-------------------------------------------------------------------%
Kp = [1, 128, 160, 176, 180, 184, 192, 255]; %Index numbers for all files
Prefix='PIDKD';PostFix='.TXT';  %Files from from the measurement

n = length(Kp);

N = 9;               %Level of wavelet decomposition, increase to smoothing
wName='db5';         %Specific wavelet name used

%Prefix='OpenLoop';PostFix='.TXT';  %Files from from the measurement

for k = 1:n
      Idx = Kp(k);         %Idex number for the filename
      Raw = load([Prefix,num2str(Idx),'.txt']);  %Load the file
```

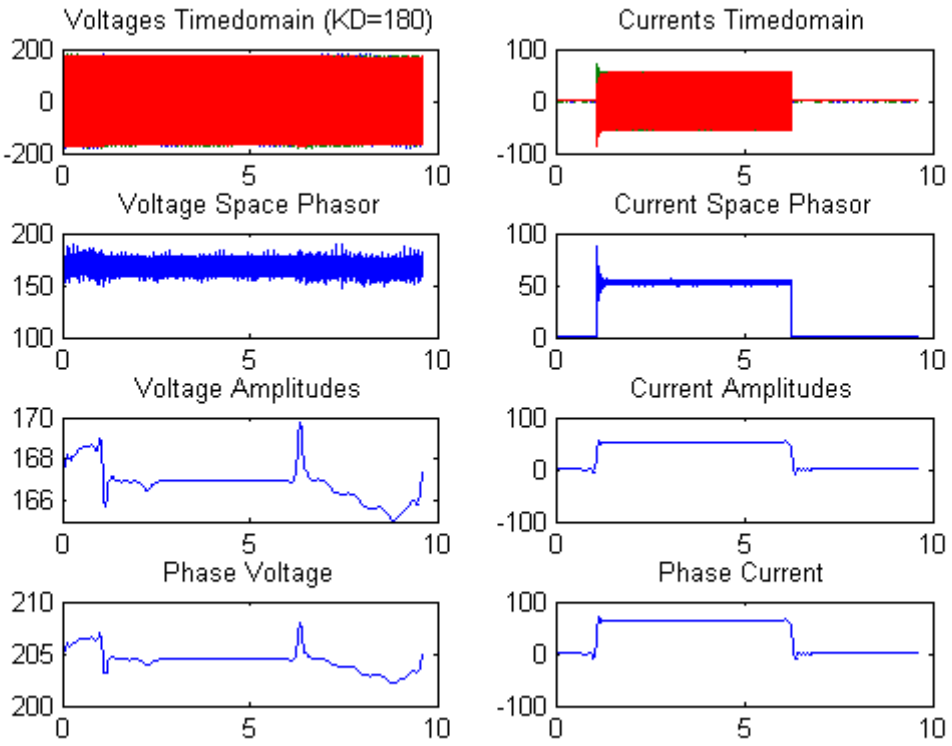
```
      t = Raw(:,1);        % First column is the time axis
      RST = Raw(:,2:4);    % R, S, T Phases
      I_rst = Raw(:,5:7);  % Current for each three phases(I_r, I_s, I_t)

    % Computing the SPACE PHASOR to find a measure for Amplitude
    % and a hard wavelet filter to remove the noise
    % Call function that compute space phsor for voltage and current
    Usp=abs(ComputeSpacePhasor(RST));   %Voltage Space Phaseor
    Isp=abs(ComputeSpacePhasor(I_rst)); %Current Space Phasor

    %Filtering the Noise from the measurements
    [C,L] = wavedec(Usp,N,wName); %Perform wavelet decomposition of Usp
    C(L(1)+1:end)=0;
    FiltUsp = waverec(C,L,wName); %Perform wavelet reconstruction of Usp
    [C,L] = wavedec(Isp,N,wName); %Perform wavelet decomposition of Isp
    C(L(1)+1:end)=0;
    FiltIsp = waverec(C,L,wName); %Perform wavelet reconstruction of Isp

          %Computing the Phase to Phase Voltage and Current
    %-----------------------------------------------------------------%
    Upp = FiltUsp * sqrt(3)/sqrt(2);
    Ipp = FiltIsp * sqrt(3)/sqrt(2);

    % Show the Corresponding Plots
    h=figure;
    subplot(4,2,1); plot(t,RST);
    title(['Voltages Timedomain (KD=',num2str(Idx),')']);
    subplot(4,2,2); plot(t,I_rst);      title('Currents Timedomain');
    subplot(4,2,3); plot(t,Usp);        title('Voltage Space Phasor');
    subplot(4,2,4); plot(t,Isp);        title('Current Space Phasor');
    subplot(4,2,5); plot(t,FiltUsp);    title('Voltage Amplitudes');
    subplot(4,2,6); plot(t,FiltIsp);    title('Current Amplitudes');
    subplot(4,2,7); plot(t,Upp);        title('Phase Voltage');
    subplot(4,2,8); plot(t,Ipp);        title('Phase Current');
end
```

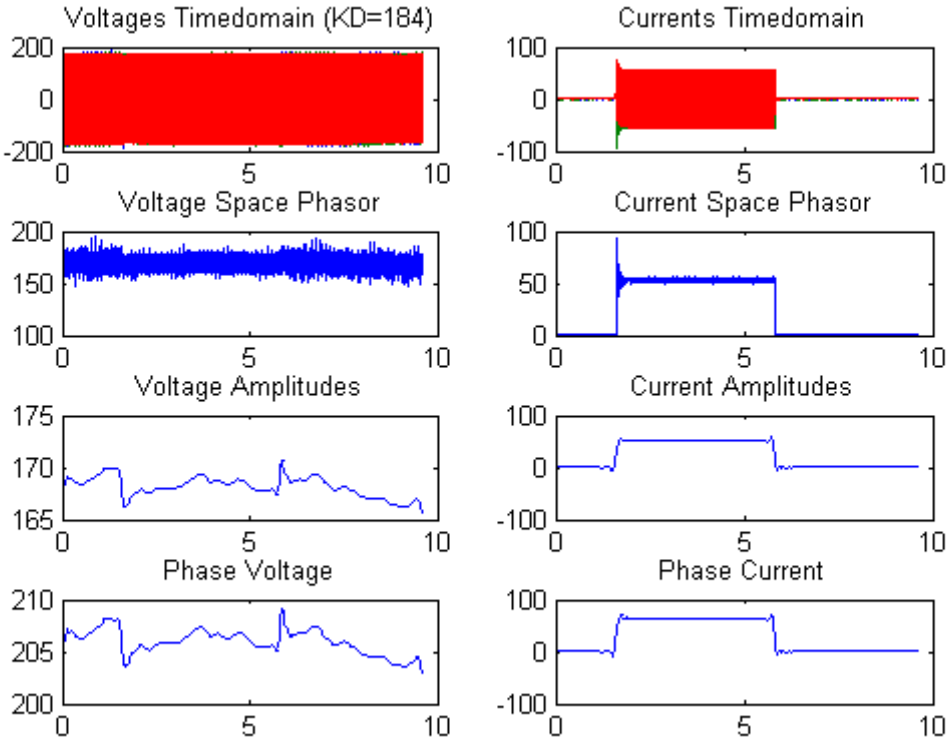

Voltages Timedomain (KD=1)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

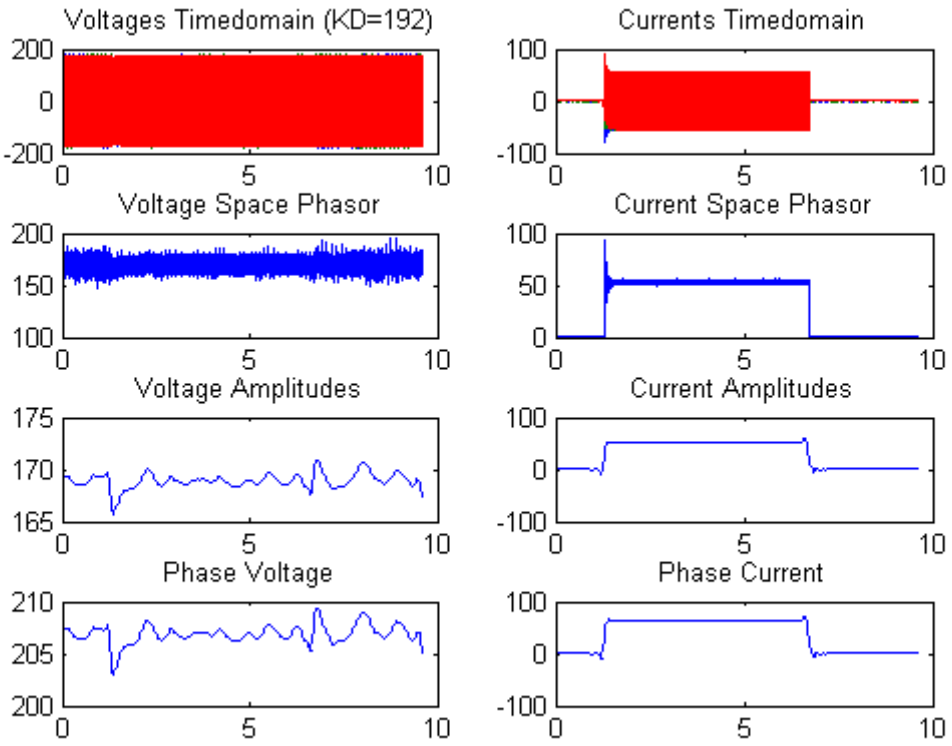

Voltages Timedomain (KD=128)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

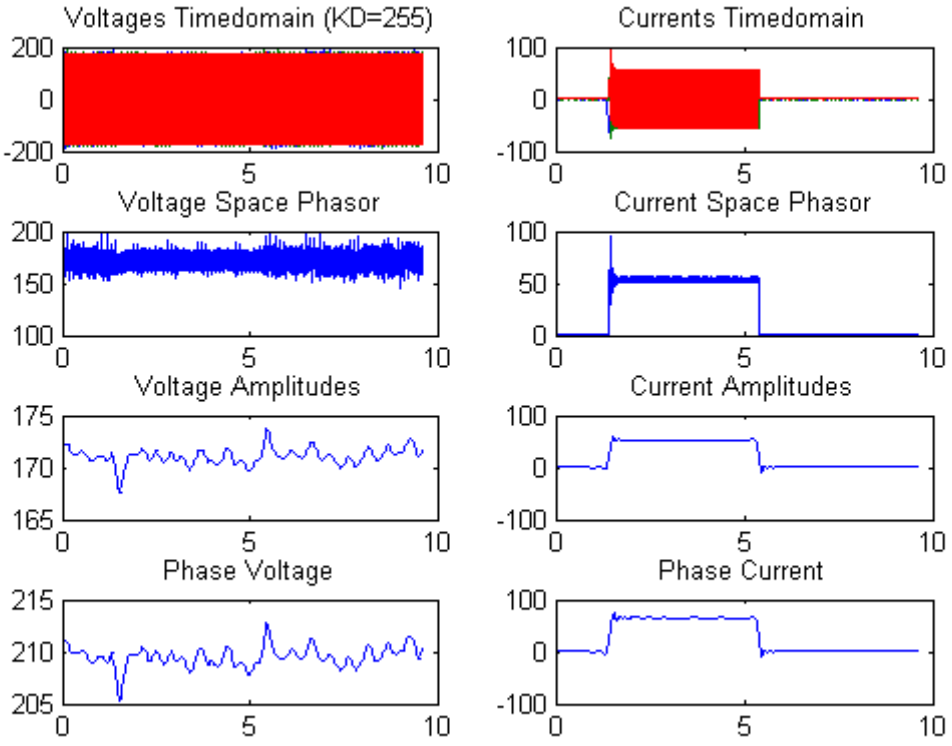

Voltages Timedomain (KD=160)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KD=176)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KD=180)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KD=184)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KD=192)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current


Voltages Timedomain (KD=255)
Currents Timedomain
Voltage Space Phasor
Current Space Phasor
Voltage Amplitudes
Current Amplitudes
Phase Voltage
Phase Current

Published with MATLAB® R2013a

```
% ######################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)           #######
% #######            SYSTEMS AND CONTROL ENGINEERING          #######
% #######          FUNCTION FOR COMPUTING SPACE PHASOR        #######
% ######################################################################


     %Function For Computing the Space Phasor
%-------------------------------------------------%
function SP = ComputeSpacePhasor(samples)
SP=2/3*(samples(:,1)+samples(:,2)*exp(1i*2*pi/3)+samples(:,3)*exp(1i*4
*pi/3));
```

15.16 APPENDIX 16: MEASUREMENT ANALYSIS

The codes for the measurement analysis are shown in the following sections.

15.16.1 SETPOINT JUMPS MATLAB CODE

```
% ######################################################################
% #######            ACHEMA HOSEA EGBUNU (142773)             #######
% #######          SYSTEMS AND CONTROL ENGINEERING            #######
% #######        Analysis of the Measurement Data             #######
% ######################################################################
close all; clear all; clc; %clear the command window, work space &
history

%%
LowSetpoint = 231;       % Phase to Phase HighVoltage setpoint
HighSetpoint = 209;      % Phase to Phase LowVoltage setpoint

          %File index number which is part of the filename
%-----------------------------------------------------------------------
%
fIdx = [1, 2, 3, 4, 5, 6];      %Index numbers for the filename
Prefix='Meas';PostFix='.TXT';  %Files from from the measurement

n = length(fIdx);

N = 9;                %Level of wavelet decomposition, increase to
smoothing
```

```
wName='db5';          %Specific wavelet name used
%%
for k = 1:n

      Idx = fIdx(k);
      Raw = load([Prefix,num2str(Idx),'.txt']);  %Load the file
      t = Raw(:,1);         % First column is the time axis
      RST = Raw(:,2:4);     % R, S, T Phases
      I_rst = Raw(:,5:7);   % Current for each three phases(I_r, I_s,
I_t)

    % Computing the SPACE PHASOR to find a measure for Amplitude
    % and a hard wavelet filter to remove the noise
    % Call function that compute space phsor for voltage and current
    Usp=abs(ComputeSpacePhasor(RST));   %Voltage Space Phaseor
    Isp=abs(ComputeSpacePhasor(I_rst)); %Current Space Phasor

                %Filtering the Noise from the measurements
    %-------------------------------------------------------------------
---%
    [C,L] = wavedec(Usp,N,wName); %Perform wavelet decomposition of
Usp
    C(L(1)+1:end)=0;
    FiltUsp = waverec(C,L,wName); %Perform wavelet reconstruction of
Usp
    [C,L] = wavedec(Isp,N,wName); %Perform wavelet decomposition of
Isp
    C(L(1)+1:end)=0;
    FiltIsp = waverec(C,L,wName); %Perform wavelet reconstruction of
Isp

    Upp = FiltUsp * sqrt(3)/sqrt(2);
    Ipp = FiltIsp * sqrt(3)/sqrt(2);

if k == 1
       %Setpoints for Voltage Amplitude and Phase to Phase Voltage
%-------------------------------------------------------------------%
    idxJump = find(t>2.0);               %Time for the jump
    idxJump=idxJump(1);
    SetpointSp = ones(size(FiltUsp,1),1); %Array to store amplitude
values
    SetpointPP = ones(size(Upp,1),1); %Array to store phase to phase
values

  %Positive Setpoint 209V to 231V (Voltage Amplitude and Phase to
Phase)
%-------------------------------------------------------------------
---%
    SetpointSp(1:idxJump)   = sqrt(2)/sqrt(3)*HighSetpoint;
    SetpointSp(idxJump:end) = sqrt(2)/sqrt(3)*LowSetpoint;

    SetpointPP(1:idxJump)   = HighSetpoint;
    SetpointPP(idxJump:end) = LowSetpoint;

    % Show the Corresponding Plots
    h=figure;
```

```
    subplot(4,2,1); plot(t,RST);
    title(['Voltages Timedomain (Measurement=',num2str(Idx),')']);
    subplot(4,2,2); plot(t,I_rst);      title('Currents Timedomain');
    subplot(4,2,3); plot(t,Usp);        title('Voltage Space Phasor');
    subplot(4,2,4); plot(t,Isp);        title('Current Space Phasor');

    subplot(4,2,5); plot(t,[FiltUsp,SetpointSp]);
    axis([0 inf, 150 240]); title('Voltage Amplitudes');
    subplot(4,2,6); plot(t,FiltIsp); title('Current Amplitudes');

    subplot(4,2,7); plot(t,[Upp, SetpointPP]);
    axis([0 inf, 150 240]); title('Phase Voltage');
    subplot(4,2,8); plot(t,Ipp); title('Phase Current');

else if k == 2
        idxJump = find(t>1.3);               %Time for the jump
        idxJump=idxJump(1);
        SetpointSp = ones(size(FiltUsp,1),1); %Array to store
amplitude
        SetpointPP = ones(size(Upp,1),1); %Array to store phase to
phase

        SetpointSp(1:idxJump)   = sqrt(2)/sqrt(3)*LowSetpoint;
        SetpointSp(idxJump:end) = sqrt(2)/sqrt(3)*HighSetpoint;

        SetpointPP(1:idxJump)   = LowSetpoint;
        SetpointPP(idxJump:end)  = HighSetpoint;

        % Show the Corresponding Plots
        h=figure;
        subplot(4,2,1); plot(t,RST);
        title(['Voltages Timedomain (Measurement=',num2str(Idx),')']);
        subplot(4,2,2); plot(t,I_rst);      title('Currents
Timedomain');
        subplot(4,2,3); plot(t,Usp);        title('Voltage Space
Phasor');
        subplot(4,2,4); plot(t,Isp);        title('Current Space
Phasor');

        subplot(4,2,5); plot(t,[FiltUsp,SetpointSp]);
        axis([0 inf, 150 240]); title('Voltage Amplitudes');
        subplot(4,2,6); plot(t,FiltIsp); title('Current Amplitudes');

        subplot(4,2,7); plot(t,[Upp, SetpointPP]);
        axis([0 inf, 150 240]); title('Phase Voltage');
        subplot(4,2,8); plot(t,Ipp); title('Phase Current');

    else if k == 3
            idxJump = find(t>1.7);               %Time for the jump
            idxJump=idxJump(1);
            SetpointSp = ones(size(FiltUsp,1),1); %Array to store
amplitude
            SetpointPP = ones(size(Upp,1),1); %Array to store phase to
phase
```

```
            SetpointSp(1:idxJump)    = sqrt(2)/sqrt(3)*HighSetpoint;
            SetpointSp(idxJump:end) = sqrt(2)/sqrt(3)*LowSetpoint;

            SetpointPP(1:idxJump)    = HighSetpoint;
            SetpointPP(idxJump:end) = LowSetpoint;

            % Show the Corresponding Plots
            h=figure;
            subplot(4,2,1); plot(t,RST);
            title(['Voltages Timedomain
(Measurement=',num2str(Idx),')']);
            subplot(4,2,2); plot(t,I_rst);  title('Currents
Timedomain');
            subplot(4,2,3); plot(t,Usp);    title('Voltage Space
Phasor');
            subplot(4,2,4); plot(t,Isp);    title('Current Space
Phasor');

            subplot(4,2,5); plot(t,[FiltUsp,SetpointSp]);
            axis([0 inf, 150 240]); title('Voltage Amplitudes');
            subplot(4,2,6); plot(t,FiltIsp); title('Current
Amplitudes');

            subplot(4,2,7); plot(t,[Upp, SetpointPP]);
            axis([0 inf, 150 240]); title('Phase Voltage');
            subplot(4,2,8); plot(t,Ipp); title('Phase Current');

        elseif k == 4
            idxJump = find(t>1.38);               %Time for the jump
            idxJump=idxJump(1);
            SetpointSp = ones(size(FiltUsp,1),1); %Array to store
amplitude
            SetpointPP = ones(size(Upp,1),1); %Array to store phase to
phase

            SetpointSp(1:idxJump)    = sqrt(2)/sqrt(3)*LowSetpoint;
            SetpointSp(idxJump:end) = sqrt(2)/sqrt(3)*HighSetpoint;

            SetpointPP(1:idxJump)    = LowSetpoint;
            SetpointPP(idxJump:end) = HighSetpoint;

            % Show the Corresponding Plots
            h=figure;
            subplot(4,2,1); plot(t,RST);
            title(['Voltages Timedomain
(Measurement=',num2str(Idx),')']);
            subplot(4,2,2); plot(t,I_rst);  title('Currents
Timedomain');
            subplot(4,2,3); plot(t,Usp);    title('Voltage Space
Phasor');
            subplot(4,2,4); plot(t,Isp);    title('Current Space
Phasor');

            subplot(4,2,5); plot(t,[FiltUsp,SetpointSp]);
            axis([0 inf, 150 240]); title('Voltage Amplitudes');
```

```
            subplot(4,2,6); plot(t,FiltIsp); title('Current
Amplitudes');

            subplot(4,2,7); plot(t,[Upp, SetpointPP]);
            axis([0 inf, 150 240]); title('Phase Voltage');
            subplot(4,2,8); plot(t,Ipp); title('Phase Current');

        elseif k == 5
            idxJump = find(t>1.6);       %Time for the jump
            idxJump=idxJump(1);
            SetpointSp = ones(size(FiltUsp,1),1); %Array amplitude
values
            SetpointPP = ones(size(Upp,1),1); %Array phase to phase
values

            SetpointSp(1:idxJump)   = sqrt(2)/sqrt(3)*HighSetpoint;
            SetpointSp(idxJump:end) = sqrt(2)/sqrt(3)*LowSetpoint;

            SetpointPP(1:idxJump)   = HighSetpoint;
            SetpointPP(idxJump:end) = LowSetpoint;

            % Show the Corresponding Plots
            h=figure;
            subplot(4,2,1); plot(t,RST);
            title(['Voltages Timedomain
(Measurement=',num2str(Idx),')']);
            subplot(4,2,2); plot(t,I_rst);  title('Currents
Timedomain');
            subplot(4,2,3); plot(t,Usp);    title('Voltage Space
Phasor');
            subplot(4,2,4); plot(t,Isp);    title('Current Space
Phasor');

            subplot(4,2,5); plot(t,[FiltUsp,SetpointSp]);
            axis([0 inf, 150 240]); title('Voltage Amplitudes');
            subplot(4,2,6); plot(t,FiltIsp); title('Current
Amplitudes');

            subplot(4,2,7); plot(t,[Upp, SetpointPP]);
            axis([0 inf, 150 240]); title('Phase Voltage');
            subplot(4,2,8); plot(t,Ipp); title('Phase Current');

        else
            idxJump = find(t>1.3);               %Time for the jump
            idxJump=idxJump(1);
            SetpointSp = ones(size(FiltUsp,1),1); %Array to store
amplitude
            SetpointPP = ones(size(Upp,1),1); %Array to store phase to
phase

            SetpointSp(1:idxJump)   = sqrt(2)/sqrt(3)*LowSetpoint;
            SetpointSp(idxJump:end) = sqrt(2)/sqrt(3)*HighSetpoint;

            SetpointPP(1:idxJump)   = LowSetpoint;
            SetpointPP(idxJump:end) = HighSetpoint;
```

```
            % Show the Corresponding Plots
            h=figure;
            subplot(4,2,1); plot(t,RST);
            title(['Voltages Timedomain
(Measurement=',num2str(Idx),')']);
            subplot(4,2,2); plot(t,I_rst);  title('Currents
Timedomain');
            subplot(4,2,3); plot(t,Usp);    title('Voltage Space
Phasor');
            subplot(4,2,4); plot(t,Isp);    title('Current Space
Phasor');

            subplot(4,2,5); plot(t,[FiltUsp,SetpointSp]);
            axis([0 inf, 150 240]); title('Voltage Amplitudes');
            subplot(4,2,6); plot(t,FiltIsp); title('Current
Amplitudes');

            subplot(4,2,7); plot(t,[Upp, SetpointPP]);
            axis([0 inf, 150 240]); title('Phase Voltage');
            subplot(4,2,8); plot(t,Ipp); title('Phase Current');
        end
    end
end
end
%%
```

15.16.2 ACTIVE POWER JUMPS FOR P CONTROL

15.16.3 ACTIVE POWER JUMPS FOR PI CONTROL

15.16.4 MATLAB CODE FOR ACTIVE, REACTIVE AND NEGATIVE REACTIVE POWER JUMPS

```
% ################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)          #######
% #######            SYSTEMS AND CONTROL ENGINEERING         #######
% #######            ANALYSIS OF THE MEASUREMENT DATA        #######
% ################################################################
close all; clear all; clc; %Close all open files, clear workspace &
history

%%
Prefix='FileName';PostFix='.TXT';  %Files from from the measurement
Idx = idxNo;      %File index number which is part of the filename:
eg:131
```

```
N = 9;              %Level of wavelet decomposition, increase to
smoothing
wName='db5';        %Specific wavelet name used

Raw = load([Prefix,num2str(Idx),'.txt']);  %Load the file
t = Raw(:,1);         % First column is the time axis
RST = Raw(:,2:4);     % R, S, T Phases
I_rst = Raw(:,5:7);   % Current for each three phases(I_r, I_s, I_t)

% Computing the SPACE PHASOR to find a measure for Amplitude
% and a hard wavelet filter to remove the noise
% Call function that compute space phsor for voltage and current
Usp=abs(ComputeSpacePhasor(RST));    %Voltage Space Phaseor
Isp=abs(ComputeSpacePhasor(I_rst)); %Current Space Phasor

%%
               %Filtering the Noise from the measurements
%-------------------------------------------------------------------%
[C,L] = wavedec(Usp,N,wName); %Perform wavelet decomposition of Usp
C(L(1)+1:end)=0;
FiltUsp = waverec(C,L,wName); %Perform wavelet reconstruction of Usp
[C,L] = wavedec(Isp,N,wName); %Perform wavelet decomposition of Isp
C(L(1)+1:end)=0;
FiltIsp = waverec(C,L,wName); %Perform wavelet reconstruction of Isp

        %Computing the Phase to Phase Voltage and Current
%-------------------------------------------------------------------%
Upp = FiltUsp * sqrt(3)/sqrt(2);    %Phase to Phase Voltage
Ipp = FiltIsp * sqrt(3)/sqrt(2);    %Phase to Phase Current

%%
% Show the Corresponding Plots
h=figure;
subplot(4,2,1); plot(t,RST);
title(['Voltages Timedomain (Active Power
JumpsKi=',num2str(Idx),')']);
subplot(4,2,2); plot(t,I_rst);      title('Currents Timedomain');
subplot(4,2,3); plot(t,Usp);        title('Voltage Space Phasor');
subplot(4,2,4); plot(t,Isp);        title('Current Space Phasor');
subplot(4,2,5); plot(t,FiltUsp);    title('Voltage Amplitudes');
subplot(4,2,6); plot(t,FiltIsp);    title('Current Amplitudes');
subplot(4,2,7); plot(t,Upp);        title('Phase to Phase Voltage');
subplot(4,2,8); plot(t,Ipp);        title('Phase to Phase Current');
```

Function: SPACE PHASOR

```
% ###################################################################
% #######              ACHEMA HOSEA EGBUNU (142773)          #######
% #######            SYSTEMS AND CONTROL ENGINEERING         #######
% #######          FUNCTION FOR COMPUTING SPACE PHASOR       #######
% ###################################################################

%%
     %Function For Computing the Space Phasor
```

```
%-------------------------------------------------%
function SP = ComputeSpacePhasor(samples)
SP = 2/3*(samples(:,1) + samples(:,2)*exp(1i*2*pi/3) +
samples(:,3)*exp(1i*4*pi/3));
```

15.16.5 APPENDIX 17: WIND TURBINE DATA ANALYSIS

```
close all; clear all;
Raw = load('20160602_16-15-10_readme.txt');   %Load data

% Get Data for Same Time Axis
t = Raw.i1_Netz_time;
iRST = Raw.i1_Netz;
x = Raw.i2_Netz_time;
y = Raw.i2_Netz;
iRST = [iRST, spline(x,y,t)];
x = Raw.i3_Netz_time;
y = Raw.i3_Netz;
iRST = [iRST, spline(x,y,t)];

x = Raw.u1N_Netz_time;
y = Raw.u1N_Netz;
uRST = spline(x,y,t);
x = Raw.u2N_Netz_time;
y = Raw.u2N_Netz;
uRST = [uRST, spline(x,y,t)];
x = Raw.u3N_Netz_time;
y = Raw.u3N_Netz;
uRST = [uRST, spline(x,y,t)];


UsedWave='db3'; WavPerc = 100*ones(1,9);

U = sqrt(sum(uRST.^2, 2));
I = sqrt(sum(iRST.^2, 2));
P = uRST.*iRST;             %Computing power


UF = MyFilter(U, UsedWave, WavPerc);
IF = MyFilter(I, UsedWave, WavPerc);
PF = MyFilter(P, UsedWave, WavPerc);

h = figure;
plot(t,uRST); title('The 3 Phases for R, S, T'); xlabel('Time');

h = figure;
plot(t,U, t,UF);
title('Voltage Space Phasor and its Filtered');
xlabel('Time');

h = figure;
plot(t,iRST);
```

```
title('Currents for R, S, T'); xlabel('Time');

h = figure;
plot(t,I,t,IF);
title('Current Space Phasor for RST and its Filtered');
xlabel('Time');

h = figure;
plot(t,P,t,PF);
title('Power for RST and its Filtered'); xlabel('Time');

%publish('Runme.m','pdf')
```

15.17 APPENDIX 18: USER GUIDE

15.17.1 HANDLING INSTRUCTION

The handling instructions indicated in this section should be followed for smooth operation of the process and program.

Human Machine Interface (Webserver):

- For the pen loop control duty cycle is used as setpoint (0 - 255). 0 duty cycle represents 0V and 255 represents 253V (phase-to-phase).
- The norminal voltage for the 60Hz machine is *220V* and the duty cycle for the nominal voltage is *54*.
- For the close loop control, the setpoint is designed to input the measured value as setpoint
- The measured value (Filter highest values) is obtained by setting the duty cycle for the open loop control and read the corresponding measured value (Filtered value) from the *System status*.
- The measured value for the nominal voltage of *220V* (Duty cycle: 54) is: 194287.
- The measured value for the voltage of *230V* (Duty cycle: 66) is: 227197.
- Under ‚*Service*‘ ---> *System status*, set the ‚*ControlTrigger Interval*‘ to value greater than 2 + Enter (for fast control).

- For a measurement electronic with no DCOffset, set the *DCOffset0* to 0 + Enter from the *System status*.
- To determine the measured value for the desired voltage setpoint, set the duty cycle for open loop for the desired voltage and read the measured value (Filter value) from the *System status*. The highest filter value represents the newest value and should be chosen. However, ensure to refresh the page using ‚*refresh values*‘ at the top of the *System status* before reading the filter value.
- Set the value of *KP, KI = 0* and *KD = 0* from (P)(I)(D) menu under *Service* for P Control.
- Set the value of *KP, KI* and *KD = 0* from (P)(I)(D) menu under *Service* for PI Control.
- Set the value of *KP, KI* and *KD* from (P)(I)(D) menu under *Service* for PID Control.
- Values of control coefficients: KP = 136, KI = 131, KD = 176.

15.17.2 TECHNICAL AND SERVICE DOCUMENTATION

Software:

- The index definitions for the variables in the main program should not be changed as this is hard encoded to the webserver. Changing this will affects the command(s) from the webserver (HMI) and program performance will be affected.
- The ADC Interrupt vector is operated in manual mode with timer 0 ISR used to start the conversion on channel 0 a fixed interval. The ADC ISR is called fetch sampled values once the conversion is completed by the hardware.
- The ADC ISR is the anchore for this project. It is responsible for fetching the conversion data from ADC Register, executes computation of actual value and control, updates the log buffer for RST phases, and set semaphore for data logging from the main loop. All this are done within the fixed interrupt frequency, however, logging operations can be interrupted by the interrupt if the logging takes more time than required.

- Avoid COM port serial print within the interrupt operated tasks as this takes more time to print values and may affects the interrupt timing.
- For future expansion of more control strategies, a platform has been created in the control function to call the any new control strategy.
- If setpoint in voltage from the HMI is desired for the close loop control, the function: *ChangePWMFrequency()* can be enabled in the main loop to address this but this will increase main loop execution time and process load due to float computations and sometime may leads to timeout for the webserver interface (HMI) due to interrupt suspending the main loop tasks for higher priority tasks. The range for the setpoint in voltage from HMI is 0 – 253V (phase-to-phase) for the 60Hz machine under consideration. However, for other project this can be extended by changing the bond implemented in the ‚*ChangePWMFrequency()*‘.
- The opto device on the excitation board has an inverted signal, therefore, the control signal from the controller is inverted before writing to the output pin connected to the excitation board.
- The coontrol signal is only written to the output pin only if there is changes from the previous value in order to keep the machine output stable. However, this has an effect, at initialization or startup the control signal is zero which drives the machine output voltage high at startup due to signal inversion by opto device on excitation board. As a result, setting a zero from HMI will not work, therefore, at startup the output pin is driven high with *analogWrite(PWMOutPin, 255)* in the void setup() to invert the signal to zero.
- KP = 136, KI = 131, KD = 176. The division factor (*shifting operator*) for: *KP >> 23, KI >> 22,* and *KD >> 22*. The shifing operator for each of the control coefficient is hard encoded in the software.

15.17.3 HARDWARE USED

- Arduino Ethernet
- Measurement Electronic Board
- Excitation Power Electronic Board
- RS232 to Serial Communication Board

- 60Hz synchronous machine

15.17.4 SOFTWARE USED

- Arduino Programming Environment
- MATLAB (for Data Analysis)
- LTspice IV (Electronic Components Simulation)
- EAGLE (for PCB Design)
- Microsoft Expression Web 4 (For Designing the Webserver)