# Synchronized Measurement of Machine Rotor Angle and Its Application

Jacques Delport

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Electrical and Computer Engineering

December 9th, 2014
Blacksburg, VA

Keywords: Rotor Angle, Synchronization, Power System Stabilizer

# Synchronized Measurement of Machine Rotor Angle and Its Application

Jacques Delport

## Abstract

The internal voltage angle of a generator is an important parameter that indicates the stability, both transient and steady-state, of the generator. This paper proposes a method of measuring and synchronizing the internal angle using a microprocessor and an optical encoder installed on the shaft of a generator. With a synchronized angle measurement, accurate stability studies and wide-area controls can be implemented. The experimental setup for measuring the rotor angle of a generator is explained in this work.

A wide-area power system stabilizer implementing the synchronized angle measurement is then investigated using a four machine, two-area system. A synchronized remote feedback rotor angle signal is included in a traditional stabilizer design. It is shown that this remote signal helps increase the stability of the system while also having the benefit of being able to be predicted accurately. This capability makes bad data detection and communication delay compensation possible.

# Table of Contents

## List of Figures

## List of Tables

## List of Equations

# 1 Introduction

Synchronous generators are an important part of the power system industry. They provide an efficient and reliable way of producing AC electrical energy for customers. The power system's stability relies heavily on the stability of these generating machines. One of the most important indicators of generators' stability during both steady state and transient conditions is the internal induced voltage angle. As such, the rotor angle is a very important parameter when considering the power system as a whole. Since the internal rotor angle measurement is not currently measured in power systems, it has to be estimated. To be able to estimate the rotor angle, state estimation as well as load flow, which requires a lot of prior knowledge of the current system, has to be used with an accurate machine model. Due to error and computational burden in these processes, the final estimate of the rotor angle could have serious errors and delays in it, leading to possible degradation of controls and analysis that hope to utilize the estimate.

The benefits from Phasor Measurement Units (PMUs) are evident in their use for state estimation and wide area analysis and control. As digital controllers get cheaper, the cost of implementing a digital measurement device on generators becomes negligible. The device presented in this paper would work similarly to PMUs and could be used to measure the rotor angle of the generator directly as well as synchronize it, which can lead to many possible beneficial applications. With the use of a high speed optical encoder and precise GPS timing, it is possible to create a highly accurate and cost effective device for measuring the angle. The IEEE C37.118 standard would make it easy to incorporate this measurement into all the current and future systems that are currently making use of the PMU technology. With this extra input, controllers have more information on which to act, making them more reliable and dependable.

## 1.1 Contributions

In this document, an accurate measurement of a synchronous generator's rotor angle has been created and synchronized. The measurement system makes use of an optical encoder to generate highly precise measurements and makes use of Precision Time Protocol (PTP) to adaptably retain accuracy regardless of oscillator drift. The synchronized angle is then used as a global signal for a local Power System Stabilizer Controller (PSS) to increase the damping of inter-area oscillations of a system exhibiting a two-area oscillation profile. The angle measurement has similar performance to speed deviations with the benefit of higher attenuation at torsional modes of oscillation. A rotor angle prediction is then created to help compensate for delays and data dropout on the controller communication channel.

## 1.2 Outline

This work documents the implementation and testing of an internal rotor angle measurement system. Chapter 2 provides a quick literature review of previous attempts to measure rotor angle. Chapter 3 provides an overview of how the encoder measures the angle as well as its setup. Chapter 4 goes over the actual experimental setup in the laboratory, the equipment used, and the software implementation of the angle measurement system. The results of the experiments are then discussed in Chapter 5. Chapter 6 then implements the rotor angle as a novel input to a power system stabilizer (PSS) along with simulation results. Chapter 7 briefly discusses some other possible applications while Chapter 8 provides some conclusions and future work. Lastly, a list of the references used in this paper is given along with Appendices of more experimental results and the code used in this work.

## 2 Literature Review

Kinitsky laid the foundation of directly measuring the rotor angle of a synchronous generator while the machine is running in 1958. The solution Kinitsky specifies is directed at a cylindrical rotor machine [1]. By using the steady state short circuit current and field current of the generator, he develops the following equation:

$$\sin(\delta) = \frac{E_t I cos(\phi)}{K E_t I_f} \tag{2-1}$$

where

$$\delta = rotor\ angle$$
$$E_t = terminal\ voltage$$
$$\phi = current\ angle$$
$$I_f = field\ current$$
$$K = machine\ constant$$

Kinitsky then uses this equation to create the setup for measuring the rotor angle. The experiment uses two Watt-meters to measure the ratio of power output to the steady-state stability limit of the synchronous machine [1]. The experiment then uses two dynamometers. The deflecting torque of a dynamometer is dependent upon the sine of the angle between the moving and fixed coils. Therefore, the maximum torque will happen when they are 90 degrees apart and minimum when they are in phase. The coils of one dynamometer are connected to the armature windings such that the torque is proportional to the output power of the machine multiplied by the sine of the deflecting angle. The other dynamometer coils are attached 90 degrees out of phase with the first one so that its torque is proportional to the terminal voltage times the field current times the cosine of the deflecting angle. Thus, taking the ratio will give a signal proportional to the ratio of power output to the stead-state stability limit of the synchronous generator, and the rotor angle can be calculated.

Kinitsky's solution had an error of 0 to 0.5 degrees when measuring the rotor angle. However, this method required very exact placement and needed very precise calibration from time to time due to the vibrations of the machine displacing the dynamometers from their original 90 degree placement. If the coils become displaced from their original position, they would no longer give a true proportional signal to the sine or cosine of the angle. This also had other drawbacks in that it was based off of cylindrical rotor machines, and its analysis was based off of steady state assumptions so that it would not work well under severe transients or salient pole machines.

D.G. Lewis and W.E. Austin then developed the next big contribution to rotor angle measurement in 1969. Rotor angle measurements before them used filters that caused unwanted slow response times [2]. They created a staircase quantized signal that used pulse-width and pulse-height conversion from a photo-electric transducer. This solution was simple and cheap and did not need to be continually calibrated. This system, however, was only useful for a real-time view of the rotor angle as there was no communication or data storage for post-mortem analysis.

There were a few attempts afterwards at measuring the speed of synchronous machines and the rotor angle position using the stroboscopic effect with digital devices [3]. However, these effectively used pulse trains from tabs placed on the shaft to obtain an average speed and angle using sequential circuits. Microprocessors were beginning to be used with similar measurement techniques as in [4].

In 1990, [5] used an optical encoder with a microprocessor, as in this work. However, the optical encoder used was incremental encoder. It could only measure speed over time with an effective pulse train created by the optical encoder. It then used a 90 kHz phase lock looped to lock on to a different reference machine fundamental to compare to this pulse train to derive the rotor angle. In 1994, de Mello found a way without using mounted devices to obtain the rotor angle [6]. De Mello measured the zero sequence voltage of the generator. He then used a derivation to prove that in synchronous machines, 3rd and 9th order harmonics can be used to compare to a fundamental frequency shift to obtain rotor angle due to the non-sinusoidal nature of the coil connections in a synchronous machine [6].

Finally, in [7] an optical encoder is used in conjunction with a Programmable Automation Controller (PAC) to measure the rotor angle and synchronize the measurement so that it may be used in local and wide-area applications as well as with post-mortem analysis. The paper takes great care to make sure the system is as time-accurate as possible. However, it does not take into account the delay from the overhead in the PAC. Furthermore, this measurement is never tested against anything other than a constant frequency source to see its accuracy. If there were any common delay to all measurements in the system, as in a PAC interrupt call delay that is not accounted for, the frequency would show to be very accurate with the delay would being subtracted out with the discrete derivative approximation. In other words, since frequency is the derivative of angle, any constant offset the angle sees will not be seen in frequency so frequency is not an all-encompassing error detection in measurements.

## 3 Rotor Angle Measurement

The torque angle of a generator is a measure of how much power is being provided. Physically, it is the angle between the rotor field and armature windings as shown in Figure 3-1 where the rotor field windings are in the direction of the d-axis.

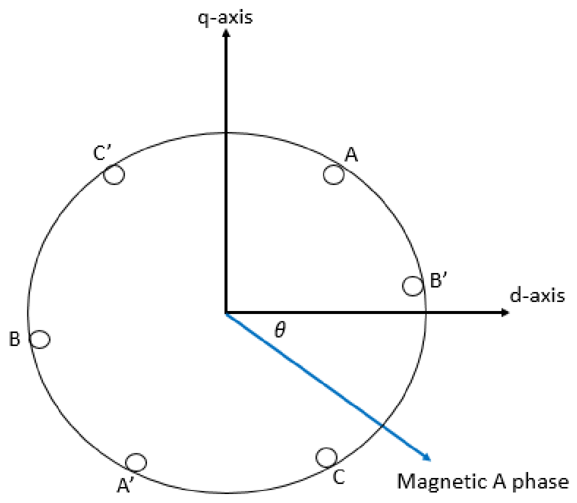

*Figure 3-1: Synchronous Machine*

Electrically, it is the angular difference between the terminal voltage and induced rotor voltage of the machine. Assuming that real and reactive power are decoupled, torque angle controls the real power flow in a generator. The torque angle ranges from 0 to 90 degrees where 90 degrees is the steady state stability limit of the generator. Since the induced voltage is internal to the machine, it cannot be measured at the terminals of the machine. Therefore, this angle cannot be measured by terminal measurements and needs to be estimated. An encoder, however, is able to measure the rotor position. Since the physical rotor position is proportional to the electrically induced angle by the number of pole pairs of the machine, the encoder can indirectly measure the electrically induced angle. The encoder is latched on to the shaft of the machine so that its optical disk spins with the coupled rotor. With the optical disk of the encoder attached to the shaft, the encoder can determine the position of the rotor at any period of time.

## 3.1 Measurement Device – Optical Encoder

An absolute optical encoder was used for measuring the rotor angle by coupling it to the rotor's shaft. The optical encoder consists of a stationary LED transmitter and photo sensor and a rotating slotted disk. When the encoder is attached to the shaft, the LED transmitter and photo sensor stay stationary while the slotted disk rotates with the rotor. The rotating disk's slots are positioned in such a way that when the transmitter shines light through it, the light forms a certain pattern in the photo sensor. The light combination read by the receiver represents a unique angular position of the encoder. Since these patterns mean nothing to the user by themselves, the encoder decodes this pattern into an understandable number. By using a reference position set to zero, the encoder sets all other encoded positions as incremental changes to the reference. The user has to define the encoder reference to decide which position should be its zero.

## 3.2 Referencing the Encoder

For angle comparison to be useful in the power system, a reference is set. Otherwise, no two angles could be compared to one another. In general, there are two types of references. This work will call them spatial and time references. Spatial referencing is when one PMU in the system is used as the reference, and all other angle measurements are compared to it. Time referencing is what is done with each individual PMU in time. 0 degrees of an electrical measurement in a PMU is defined as the maximum of the sinusoid occurring on the second rollover expressed in UTC time [8]. In other words, a PMU measuring a voltage of angle zero corresponds to Phase A experiencing its maximum on the second. For the rotor angle measurement to have any bearing on the rest of the measurements on the power system made by PMU's, a position has to be found to line up the encoder's reading with one of the phases of the voltages and currents on the synchronous machine's stator. Otherwise, there would be an inherent phase degree difference between estimated internal rotor angles and their measured counterparts. Phase A is the most obvious choice since it is the reference for PMUs with the other two phases being 120 degrees apart. To align the encoder to phase A, a little understanding about a synchronous machine's build and magnetic field positions is required.

As shown in Figure 3-1, phase A's magnetic field lags phase A by 90 degrees in the spatial domain when considering a rotor moving counter-clockwise. By definition, the exciter field windings' magnetic field is in the direction of the d-axis. Both of these directions can easily be verified with the Biot-Savart Law with the right hand rule. It can therefore be seen that to line up the encoder's reading with the phase angle of phase A, the rotor's d-axis should be lined up with phase A's magnetic field. This is done by disconnecting the machine from the system and applying a constant DC voltage across Phase A of the machine which creates a constant magnetic field in the direction of phase A. When a DC current is then applied to the exciter, a torque will develop that will align the magnetic fields of the exciter and Phase A as given by:

$$\underline{\tau} = \underline{\mu_{coil}}\ x\ \underline{B} \tag{3-1}$$

where $\mu$ is the magnetic moment of the exciter coil and $B$ is the magnetic field density of the field applied to the terminals. $\mu$ is a function of the number of coils in the exciter, the current flowing through it, and its cross-sectional area as defined in equation 3-2.

$$\underline{\mu} = NIA \tag{3-2}$$

To produce a constant voltage in line with phase A, the neutral of the machine is needed. If the neutral of the machine is not accessible, an equivalent neutral can be made. This can be done by shorting out the terminals of Phase B and Phase C, and then applying the negative of the voltage source to that new connected "neutral." This can be seen in the figure below.

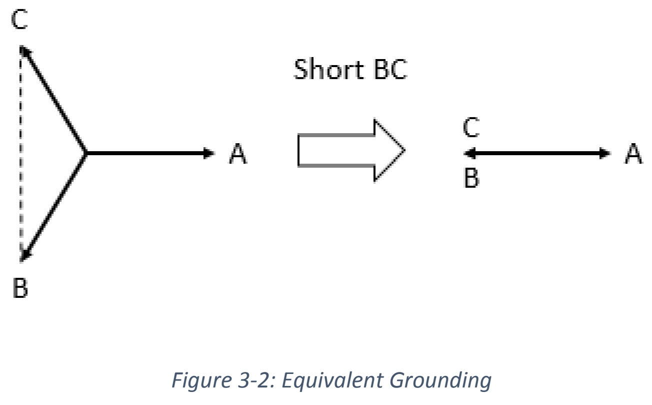

*Figure 3-2: Equivalent Grounding*

Since the excitation field is in line with the D-axis, the torque between the two constant magnetic fields will pull the rotor in line with Phase A's magnetic field. Once this is done, the encoder can then be zeroed out as a reference in that position. The optical encoder in this work is an absolute rotary encoder. This means that the reference is a mechanical positioning, and it will maintain the reference throughout the future.

## 3.3 Time-tagging the Rotor Angle

Once the encoder is referenced to phase A, it is ready to measure the rotor angle. However, for this value to be used in a synchronized manner, the rotor angle needs to be time-tagged at the moment in time that it was measured. Just like PMU's, this particular measurement system uses GPS NMEA standard timing to its advantage. Each time a measurement is made by the encoder, the measurement is time-tagged with GPS time. However, GPS NMEA only sends the time once every second on second rollovers. The time-tags for consecutive measurements after the second rollovers have to be extrapolated by the clock sampling period.

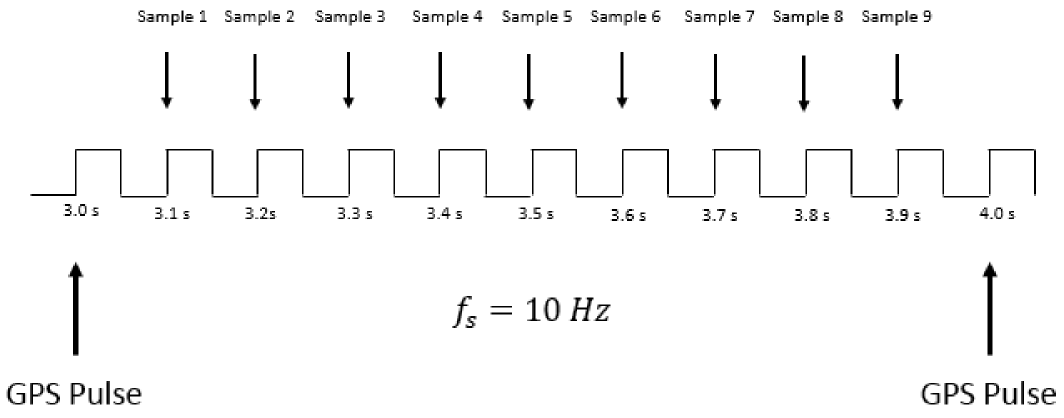

*Figure 3-3: 10 Hz Sampling Timing*

Figure 3-3 shows the extrapolation used for a 10 Hz sampling frequency. A timer is set in the microprocessor as soon as the first GPS pulse is seen, the value in the timer set corresponds to the sampling period, 0.1 seconds in this case. When the timer expires, the 0.1 seconds have passed, and the encoder's current position needs to be read as well as the timer being reset to 0.1 seconds. However, the clock accuracy is not guaranteed to be nearly as accurate as GPS timing. Therefore, extra consideration has to be taken to ensure that the corresponding measurements between GPS pulses have similar time precision. This will be discussed in further detail in Section 4.

# 4 Experimental Setup

## 4.1 Introduction

The rotor angle measurement system was tested on a small 0.25 horsepower generator to see how well it could track the generator’s angle. To be able to test whether the optical encoder system measured and reported the angle accurately, it needed to be compared against something with an established accuracy. The choice made was a Phasor Measurement Unit (PMU) at the terminals of the machine to be measured. PMUs have well established standards and accuracy requirements that could be used to benchmark this system [8]. Figure 4-1 shows the machine, encoder, and PMU setup. The figure shows the encoder on the prime mover side of the generator. However, it is actually on the other side as there is no connection to place it on the prime mover side.

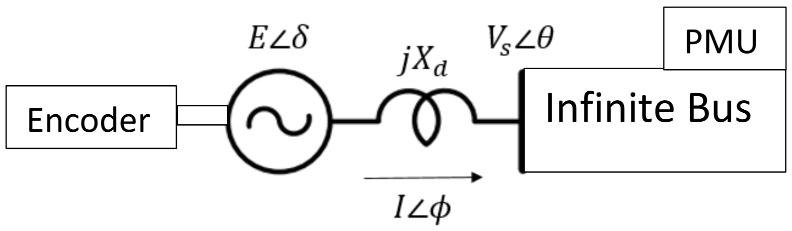

*Figure 4-1: Machine Setup*

A motor powered by a DC source was used as an equivalent prime mover for the generator. Another DC source was used as the voltage source for the exciter of the synchronous machine. A three phase source powered by the power grid in Blacksburg, VA was used as the infinite source. The machine could have been plugged straight into the grid, but the three phase source was used for safety reasons. Since the generator used is a 0.25 HP machine, the equivalent system it sees is much larger in capacity and won’t be affected by the generator. The grid can therefore be thought of as an infinite bus during any of the generator transient time periods.

To synchronize the generator to the system, the power to the DC motor was increased until the generator’s rotor was rotating at approximately the system frequency. This frequency was verified with a stroboscope tuned to the system frequency as well as lamps attached to the terminals of the machine. This had the effect of making the shaft appear immobile and the lamps luminescence remain constant once it was moving at the system frequency. Once up to speed, the exciter voltage was then increased until the terminals of the machine showed rated voltage, 208 $V_{LL\text{-}RMS}$ in this case. The lamps connected at the terminals of the generator were then used to check that the phase and sequence matched the system. If the lamps on the three phases had identical luminescence, the connection sequence was correct. When the lamps were completely unlit, it meant the phase of the terminal voltage of the generator matched that of the infinite bus, and no current was flowing. At this point, requirements for synchronism were met; the switch controlling the lamp bypass coil could be thrown to connect the generator to the system. No current is flowing through the infinite bus, and the generator is producing no power as its voltage magnitude and angle are the same as the infinite bus. If it is taken into account that the machine is a 4 pole machine, the mechanical angle should then give the electrical internal angle induced which, ignoring losses of the machine, should be equal to the generator terminal voltage angle.

The general setup for the measurement system to measure the rotor angle is shown in Figure 4-2.

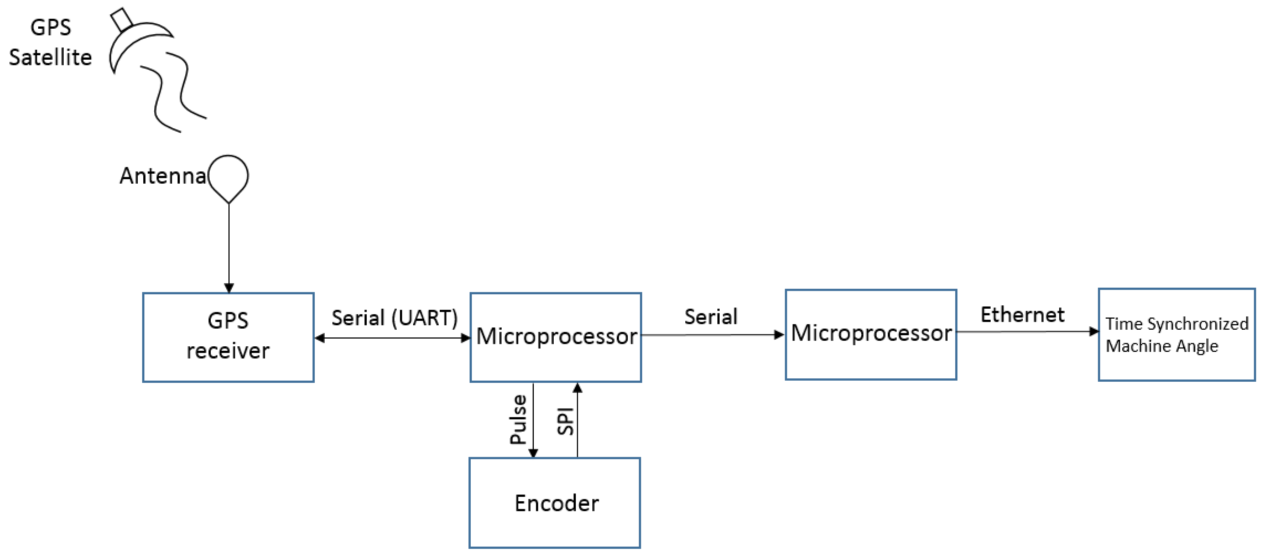

*Figure 4-2: System Layout*

On the second rollover, the GPS signal from the satellite sends a pulse. The receiver synchronizes to this pulse as will be discussed in the timing section 4.4.1. The microprocessor waits for the pulse and tells the encoder to latch and send the current value of position. Once the microprocessor has the word containing the current position, it time tags it, puts it in C37.118 format and sends it through a UART serial standard to another microprocessor which will then send that word to the network through the UDP Ethernet protocol. The second microprocessor is needed to manage Ethernet so that it will not affect the timing of the first microprocessor. The Ethernet code has a lot of interruptions and overhead that would delay the latching of the encoder value at the proper time thereby creating further error in the angle measurement.

## 4.2 Components

The components needed for the experiment were an encoder to measure the machine angle, a microprocessor to decode the encoder measurement, a GPS satellite antenna as well as a receiver for time synchronization, a machine to produce a machine angle as well as all the sources needed to run the machine, and a three phase source to mimic the power system.

### 4.2.1 Encoder

The encoder chosen was a 5 Volt Transistor-Transistor-Logic Output single-turn absolute rotary optical encoder with a 500 kHz frequency response and a latching capability. This type of encoder was chosen due to its high accuracy and reporting rate ability. With 15 bits, this encoder gives three bits for CRC error checking on the data itself and leaves 12 bits for actual data [9]. This gives a resolution of:

$$\frac{1}{2^{12}} * 360° = 0.088°.$$

It was decided this resolution was enough for testing and application purposes. However, the incremental cost of going up to an 18 bit encoder is negligible and would give a resolution performance of:

$$\frac{1}{2^{15}} * 360° = 0.01°.$$

The encoder is an absolute encoder. This means that its reference is always set to a certain position. If power to the encoder is lost, or the position is changed, the encoder will not be affected and will read the correct angle as long as it was set properly initially. Its sampling period is guaranteed to be less than 1 microsecond [9]. This corresponds to a sampling frequency of 1 MHz. However, the encoder communicates serially through a quasi-standardized SSI format using differential signals. The maximum reporting rate of this serial connection is 500 kHz. In this project, it has been set to 200 kHz, as there are no frequencies of interest nearly that high in the power system. Doing the calculations, 200 kHz transmission with an 18 bit word results in a true angle reporting rate of 11.1 kHz. Using the Nyquist sampling theorem, this means that the highest frequency dynamics that this reporting rate will capture is 5.55 kHz. For power systems, the highest frequencies of interest are below 1 kHz; with generator rotor angles in particular, the frequency range of interest is sub-synchronous, or below 60 Hz.

### 4.2.2 Microprocessor

The author of [7] attempts to measure rotor angle using PAC's to time synchronize the data. This method is extremely expensive, and while they considered the accurate timing of the clocks, they did not consider the latency introduced by the actual PAC. To measure rotor angle, a microprocessor is a sufficient option. This option is cheap, with a cost magnitude orders lower than PLCs or PACs, is easy to program, and gives complete control of latency since there is no overhead. The processor chosen for this work was the Digilent Cerebot 32MX7.

The processor timing is split into three sections: oscillator, main bus clock, and peripheral bus clock. The oscillator is a regular voltage-controlled oscillator set to operate at 3.3V at 4 MHz with a frequency stability of 100 parts per million (ppm) [10]. The frequency stability indicates that the oscillator will be, at most, 100 oscillations off for every one million oscillations. This number is different for every oscillator but remains relatively constant for each oscillator under constant operating conditions. For this reason, it can also be thought of as the drift of the oscillator. The board itself uses a phase lock loop (PLL) to create an effective 8 MHz oscillator. This means the drift for the oscillator the board sees has a maximum drift of 800 cycles per second. The board's main and peripheral bus clocks are phase-locked afterwards and are very flexible to be set to different values. The clock rates of both have been set to their maximum speeds of 80 MHz and 10 MHz, respectively, to get the best timing accuracy results. Since all these clock rates are phase locked to the original oscillator, they will result in drifts of 8000 and 1000 cycles per second, respectively.

As this project was set to use the core timer of the Cerebot, which uses the main bus's clock, the resultant accuracy of the drift is calculated as follows. At a drift of 8000 cycles per second, the following drift is seen:

$$Drift\ (seconds) = \frac{8{,}000}{80{,}000{,}000} = 0.0001\ sec = 0.1\ ms$$

Within that time, a machine spinning at 3600 rpm will have rotated:

$$3600\frac{rev}{min} * \frac{1}{60}\frac{min}{sec} = 60\frac{rev}{sec} * 360\frac{deg}{sec} = 21600\frac{deg}{sec} * 0.0001\ sec = 2.16°$$

This inaccuracy is too large. To compensate for this, a new more expensive crystal oscillator with less drift must be installed in or software must make up for this drift. An adaptive precision time protocol (PTP) algorithm was adopted (see section 4.4.2) to take advantage of the fact that the drift remains relatively constant from one minute to the next. This algorithm decreased the drift on the main bus to around 100 to 200 cycles per second. This was measured using the timer reader functions on the microprocessor and at 80 MHz corresponds to a maximum drift of approximately 2.5 ppm. Doing the same math as above, this gives a maximum error of 0.054 degrees between GPS pulses. The speed of a synchronous machine is governed by the equation

$$N_s = \frac{120f}{P}. \qquad (4\text{-}1)$$

where

$$f{:}\ nominal\ frequency$$
$$P{:}\ number\ of\ poles\ per\ phase$$
$$N_s{:}\ Synchronous\ speed$$

Since the machine in this experiment was is 4 pole machine, its speed is 1800 rpm, resulting in a maximum error of 0.027 degrees for the last timer-based encoder reading before the next GPS pulse.

Another thing of note is that the microprocessor input/output pins are single-ended pins. This means that their grounds are tied to the ground of the microprocessor. The encoder uses a differential signal so that both the signal high and signal low are electrically floating with respect to analog ground providing superior common-mode noise rejection. As the encoder will be placed on the shaft of a synchronous generator with extreme 60 Hz electromagnetic interference, this is necessary. However, for the microprocessor's clock to control the encoder, its output needs to be converted to a differential signal. This was done by sending the signal through a transistor switch acting as a NOT gate for the differential low signal and sending it through another NOT gate transistor switch for the differential high signal. This created an equivalent differential signal from the single-ended signal.

### 4.2.3 GPS Equipment

To get the GPS time synchronization, an antenna as well as a receiver were required. The antenna was a Trimble 57681-20 with an L1 frequency band of 1.57542 GHz. The GPS receiver chosen was the Gms-u1LP. This is an add-on receiver that fits the Cerebot's pin configuration form factor for the general purpose input/output (GPIO) pins. The receiver throughputs the pulse per second (PPS) from the GPS and decodes the time. It has a guaranteed PPS throughput of 1 ns and uses serial communication [11]. It was decided that this would be sufficient for the purposes of time-tagging the synchro-angle measurement.

## 4.3 GPS Timing

To be able to properly time-tag an encoder value, it is necessary to understand how the GPS timing signal works. On every second rollover, a 1-PPS is generated, and in many receivers is accurate to within a few nanoseconds of true UTC time. This pulse is sent by the GPS so that any receiver can synchronize to it and know that it is now locked on to GPS time. After the pulse is generated, data is produced that tells the user the UTC time associated with the pulse.

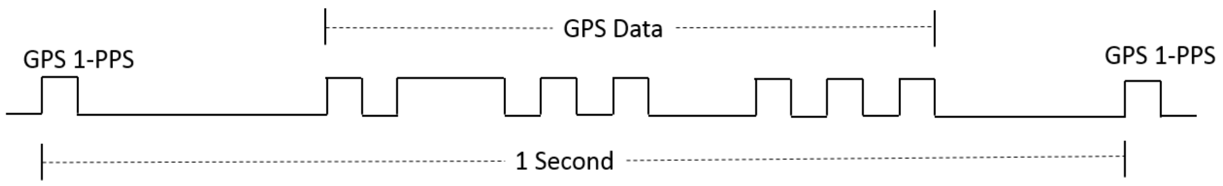

*Figure 4-3: GPS Communication*

The GPS data sent after the 1-PPS is decoded in the NMEA standard by the Gms receiver used. The receiver provides 5 different NMEA standard output sentences: GGA, GSA, GSV, RMC, and VTG. All the sentences are comma-separated value (CSV) sentences. However, only the GGA and RMC are useful to synchronization as only their output sentences provide the UTC time. The specifications for GGA and RMC are given in the tables 3-1 and 3-2 from [11]. Some of examples of the output words are as follows.

$GPRMC,064951.000,A,2307.1256,N,12016.4438,E,0.03,165.48,260406, 3.05,W,A*55

$GPGGA,064951.000,2307.1256,N,12016.4438,E,1,8,0.95,39.9,M,17.8,M,,*65

The microprocessor needs to lock on to either $GPRMC or $GPGGA, and the value afterwards will be the UTC time.

*Table 4-1: GGA NMEA Specifications*

| Name | Example | Units | Description |
|---|---|---|---|
| Message ID | $GPGGA | | GGA protocol header |
| UTC Time | 064951.000 | | hhmmss.sss |
| Latitude | 2307.1256 | | ddmm.mmmm |
| N/S Indicator | N | | N=north or S=south |
| Longitude | 12016.4438 | | dddmm.mmmm |
| E/W Indicator | E | | E=east or W=west |
| Position Fix Indicator | 1 | | See **Table-3** |
| Satellites Used | 8 | | Range 0 to 14 |
| HDOP | 0.95 | | Horizontal Dilution of Precision |
| MSL Altitude | 39.9 | meters | Antenna Altitude above/below mean-sae-level |
| Units | M | meters | Units of antenna altitude |
| Geoidal Separation | 17.8 | meters | |
| Units | M | meters | Units of geoid separation |
| Age of Diff. Corr. | | second | Null fields when DGPS is not used |
| Checksum | *65 | | |
| <CR> <LF> | | | End of message termination |

*Table 4-2: RMC NMEA Specifications*

| Name | Example | Units | Description |
|---|---|---|---|
| Message ID | $GPRMC | | RMC protocol header |
| UTC Time | 064951.000 | | hhmmss.sss |
| Status | A | | A=data valid or V=data not valid |
| Latitude | 2307.1256 | | ddmm.mmmm |
| N/S Indicator | N | | N=north or S=south |
| Longitude | 12016.4438 | | dddmm.mmmm |
| E/W Indicator | E | | E=east or W=west |
| Speed over Ground | 0.03 | knots | |
| Course over Ground | 165.48 | degrees | True |
| Date | 260406 | | ddmmyy |
| Magnetic Variation | 3.05,W | degrees | E=east or W=west **(Need GlobalTop Customization Service)** |
| Mode | A | | A= Autonomous mode<br>D= Differential mode<br>E= Estimated mode |
| Checksum | *55 | | |
| &lt;CR&gt; &lt;LF&gt; | | | End of message termination |

## 4.4 Code Procedure

The code structure that has been created for this project can be split into three mains parts: timing, communicating with the encoder, and sending out encoder data. The main logic of the code is shown in Figure 4-4.

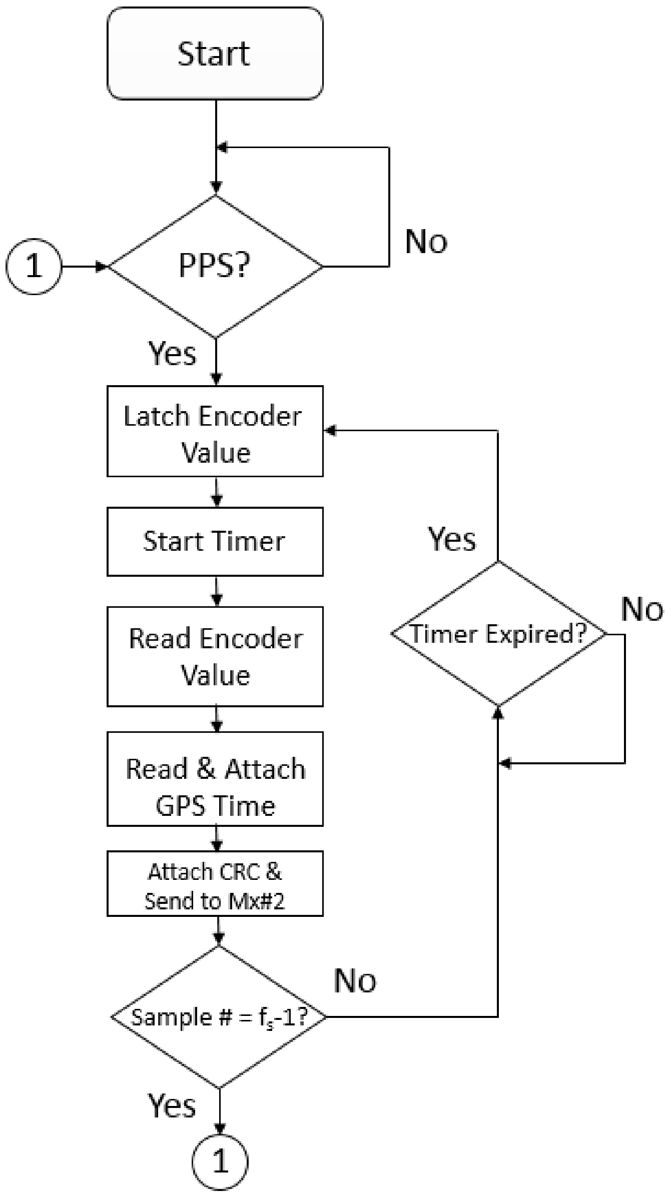

*Figure 4-4: Code Algorithm*

### 4.4.1 Precision Time Protocol (PTP) Clock Algorithm

If an oscillator is drifting from true time, its time can eventually become far off from the actual time. This can be seen in any analog clock whose time has to be reset constantly to keep in on time as seen in Figure 4-5.

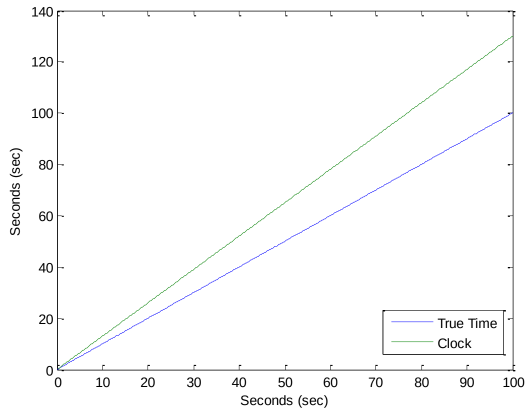

*Figure 4-5: Uncorrected Clock*

To help correct this problem, the GPS PPS can be used to bring to the clock back to true time every second as was talked about in the previous section. A visual example can be seen in Figure 4-6.

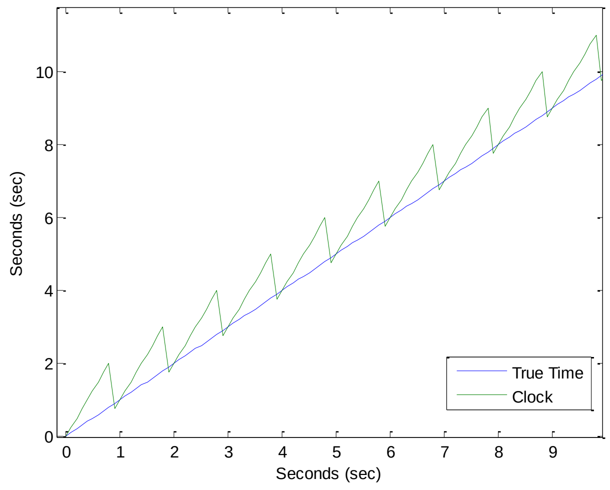

*Figure 4-6: PPS Corrected Clock*

Since the clock drift is relatively constant, however, we can increase the timing accuracy even further. Clock drift being constant means that the slope of its line can be determined. If the slope of this line is known, the angle between the two times can be gotten. One measurement of this is not good enough since the clock also has a random Gaussian jitter error associated with it. For this project, the PTP algorithm takes 10 minutes worth of samples or 600 samples to try to mitigate this error. Once this is done, the frequency of the oscillator can be fixed accordingly. If this were a temperature or oven-controlled oscillator, the temperature could be changed to change the frequency of the oscillation. Since this was a voltage controlled oscillator with a constant voltage, this oscillator's frequency could not be changed. Instead the software changes the number of cycles it should see to expect a new measurement. If the clock is drifting too fast or too slow, the software will increase or decrease the cycles expected between measurements, respectively. This algorithm reduced the drift error that the encoder would see by a factor larger than 100.

### 4.4.2 Timing

It was decided at first to use a 10 Hz sampling frequency of the encoder. This required 10 samples per second. Since GPS was being utilized, the PPS that is sent with GPS could be used to align the first sample of every second. It was decided for this project that a polling routine waiting for this PPS would be better than an interruption routine alternative. This was because time accuracy was the main concern. With polling, the microprocessor will react to the pulse much quicker since that is microprocessors main concern. Interruption routines have overhead that have to be worried about. While this overhead is miniscule for most projects, for this one it was decided to poll instead. There are ways to get around the delay from the overhead of the interrupts since interrupt overheads are constant clock cycle delays; the delay in the interruption call can be included in the timer to avoid the delay. Once the bit has been seen, the encoder is told to latch with the microprocessor SPI.

The moment the SPI is enabled until the first bit is sent is approximately 0.1 microseconds. Since the microprocessor is polling, there is approximately 10 clock cycles of latency at maximum due to the comparators and the GOTO statements in the actual assembly code. At 80 MHz, 10 clock cycles reports to around 0.125 microseconds. As soon as the encoder sees the first bit, it latches on to its current word. [9] guarantees a latency error of much less than 1 microsecond for the encoder. Making the assumption that much less means a tenth, error from the encoder is 0.1 microseconds as well. The GPS receiver has a guaranteed PPS throughput of 0.001 microseconds. Adding these latency errors together, an approximate latency error of 0.326 microseconds is obtained. This error remains constant throughout the time and are not associated with any drift. Table 4-3 shows the accumulation of error throughout the samples within one second. It assumes that the sampling frequency is 10 Hz. On the 11th sample, the second rollover has occurred, and the clock has been brought back to GPS time. One thing of note is that it is assumed that GPS time is true time and has no errors associated with it. This is solely to demonstrate the timing errors in this system. Realistically, GPS typically has a couple nanoseconds of error associated with it, so this would have to be added to the overall timing accuracy of the system.

*Table 4-3: Overall Timing Errors*

| Sample Number | Oscillator Error (micro-seconds) | Rx Error (micro-seconds) | SPI / Encoder Error (micro-seconds) | Encoder Error (micro-seconds) | Poll Error (micro-seconds) | Total Time Error (micro-seconds) | Total Degree Error (degrees for 3600 RPM) |
|---|---|---|---|---|---|---|---|
| **1st Sample** | 0 | 0.001 | 0.1 | 0.1 | 0.125 | 0.326 | 0.007 |
| **2nd Sample** | 0.25 | 0.001 | 0.1 | 0.1 | 0.125 | 0.576 | 0.012 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |
| **10th Sample** | 2.25 | 0.001 | 0.1 | 0.1 | 0.125 | 2.576 | 0.056 |
| **11th Sample** | 0 | 0.001 | 0.1 | 0.1 | 0.125 | 0.326 | 0.007 |
| ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ | ⋮ |

### 4.4.3 Communicating With the Encoder

The encoder used the communication protocol Synchronous Serial Interface (SSI) to transfer rotary information along. The encoder waits until it sees a high to low transition on the communication line to store its current position in its parallel register. The proceeding clock cycles on the communication line then drive the encoder to send bits down the line. This is similar to regular Serial Peripheral Interface (SPI) communication protocols where the master's clock drives the slave's communication. The Cerebot's SPI is configurable and was changed to match this communication protocol so the two could communicate.

The encoder also communicates in gray code [12]. Gray code is different from regular binary code in that only one bit changes from one decimal number representation to the next. This means that the maximum reading error is one step. If a bit goes corrupted, the reading will only be off by one decimal interval. This value needed to be decoded back into binary before converting it into a decimal number to be sent off.

The conversion between gray code and binary is not computationally intense and can be done using an iterative serial conversion.

### 4.4.4 Sending Data

Once the time and the angle have been gotten, the value is sent via serial to another microprocessor that handles the Ethernet to send it over UDP. To make the integration of the angle with the current system easy to integrate, the angle measurement is sent as a synchrophasor packet according to the IEEE C37.118 standard. The standard for the data packets are as follows:

| SYNC | FRAMESIZE | IDCODE | SOC | FRACSEC | DATA | CHK |
|---|---|---|---|---|---|---|
| 2 bytes | 2 bytes | 2 bytes | 4 bytes | 4 bytes | 4x(# angles) | 2 bytes |

An example of a frame of this type sent by this microprocessor is:

**AA00-0010-0001-5457A5CC-000592E4-0000-00017918-0000-0000-0000-0000-001A.**

The data packet is split into stat, phasors, freq, dfreq, analog, and digital. More about each of these unique identifiers can be read in [14]. What is important to the discussion in this paper is the SOC, FRACSEC, PHASORS, and the CHK. Everything else is easily set up by following the standards. SOC and FRACSEC are the number of seconds since January 1st, 1970 and the fraction of the second, respectively. In this paper, while the C37 standard has been followed, the time has been given in UTC instead of UNIX. This would need to be changed to create a true C37 word. FRACSEC is given as a large integer with its divisor given in the configuration frame so that the receiver knows what to divide FRACSEC by. PHASORS is a 4 byte section that in this particular project is given in polar (2 bytes for the magnitude and 2 bytes for the angle). The magnitude has been set to 1 as it never changes.

CHK is the Cyclic Redundancy Check (CRC) at the end of the frame to help distinguish if the data is corrupt or not. CRC works by taking the word meant to be sent and dividing it by some polynomial. The remainder of this division is then transmitted along with the word. On the other end, the remainder is added to the word and the resultant word is divided by the same polynomial. If the remainder is then zero, the word has most likely not been corrupted. CHK is more specifically a CRC-16-CCITT CRC. Its polynomial is expressed as

$$f(x) = x^{16} + x^{12} + x^{5} + 1 \tag{4-2}$$

Microchip Technology, Inc. makes a template for Ethernet code that had to be fixed and debugged to suit the purpose of this project. This code will not be discussed and can be found from their site.

# 5 Experimental Trials and Results

## 5.1 Infinite Bus Test

For the infinite bus experiment, the synchronous generator was connected as in Section 3. The prime mover was tuned until the generator power production was zero to make the induced rotor angle match the terminal voltage angle. A SEL-421 PMU was connected at the terminals of the generator so that the validity of the encoder measurement could be ascertained.

Figure 5-1 displays the two angles plotted in real-time over a 2 hour time period. The y-axis is the unwrapped phase of both the encoder's induced voltage angle and the PMU's terminal angle with respect to time. It can be seen from this graph that there is a difference of about seven to eight degrees between the two measurements. This is a result of referencing error. When the machine was aligned in the beginning to set the encoder's reference angle to Phase A's reference angle, there was approximately four or five degrees of play on the shaft of the generator. Since this is a four pole machine, this corresponds to an error in electrical degrees of about eight to ten degrees. Consideration was taken to make this difference zero after the measurements were taken. However, this would not be a realistic scenario. If this system were to be implemented on generators in the field, careful consideration would have to be taken to set the reference correctly.

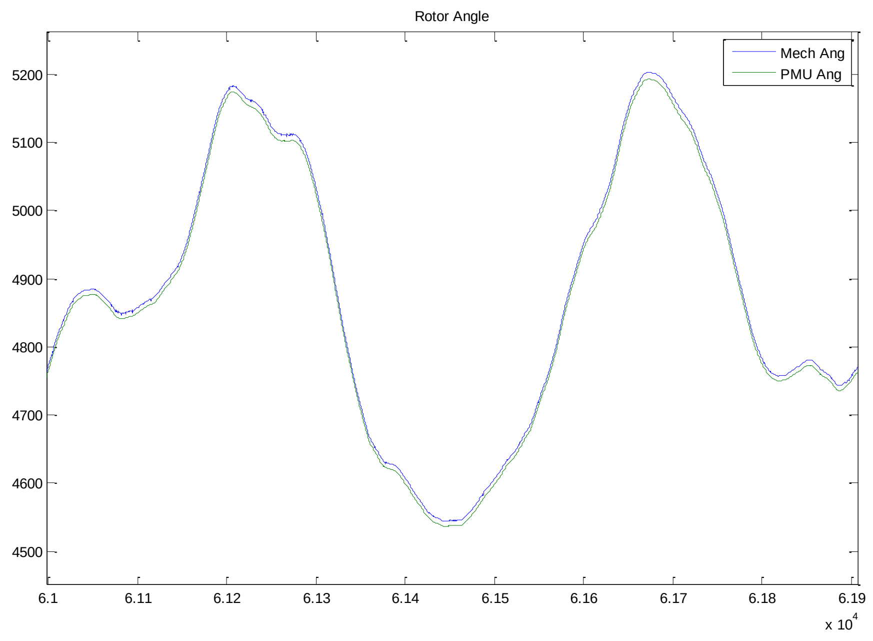

*Figure 5-1: Rotor Angle vs Terminal Angle*

Other than the constant reference error, it can be seen from Figure 5-1 that the encoder's angle follows the PMU's terminal angle calculations extremely well over the two hour test despite the frequency of the system not being nominal. To obtain this encoder angle, no prior knowledge of the system is needed. No prior studies have to be done to obtain the torque angle.

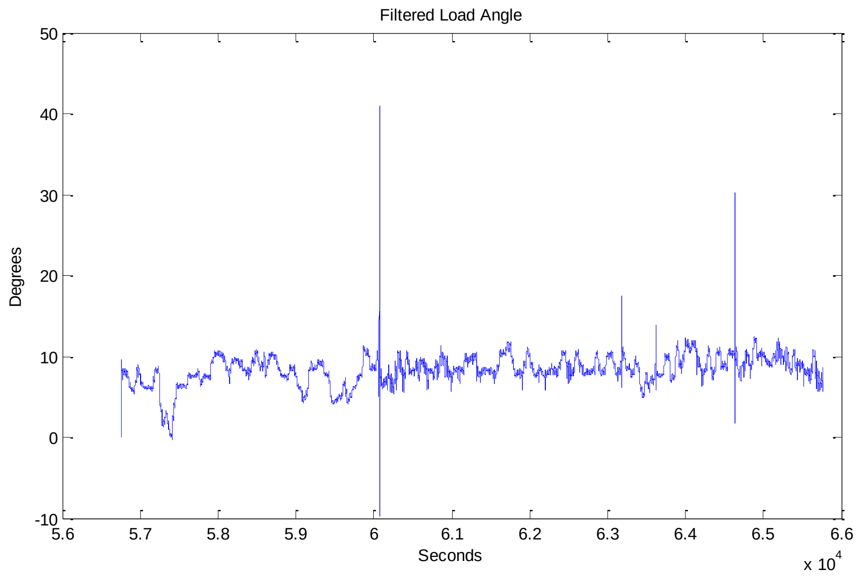

*Figure 5-2: 2 Hour Load Angle*

Figure 5-2 shows the filtered difference between the PMU measurement and the encoder measurement. It may appear that there are sudden transitions, but those transitions occur on the timescale of 10-20 seconds. These changes are most likely from the DC power supply powering the prime mover or the DC power supply providing excitation voltage not remaining constant. Any change in input mechanical power from the prime mover will change the induced angle to produce or absorb real power as is needed. However, assuming constant input power, a change in excitation voltage will result in a change induced angle to compensate for power output. For example, a synchronous generator with constant input power that experiences a loss in excitation voltage will experience an increased induced angle as seen in Equation 5-1.

$$P = \frac{EV_s}{X_d}\sin(\delta) \qquad (5\text{-}1)$$

Small changes in current were measured on the terminals of the generator. This makes the author believe that these were real changes in induced angle versus terminal angle and not measurement errors. The noisy spikes in the graph are explained by packet loss over Ethernet from the PMU and encoder for those time periods.

Figure 5-3 plots a filtered zoomed in graph of Figure 5-2. The encoder has captures a power oscillation of the machine while it was running. This frequency of oscillation is around 0.29 Hz. As this frequency is so low, it is believed that it is not the machine oscillating against the power system. A small machine of this size has much smaller inertia time constants and would more likely produce oscillations in the range of 1-3 Hz. It is instead believed to be an inter-area oscillation or even a global oscillation of the entire system that the PMU did not capture. This can be explained by a strong power system at the time of measurement as will be discussed in Section 6.1.2.

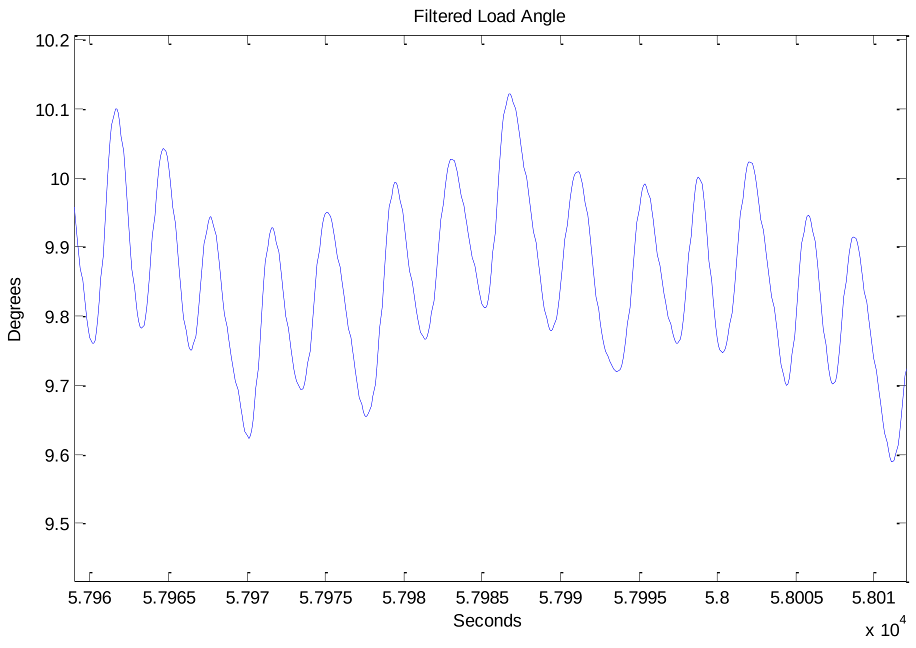

*Figure 5-3: Power Oscillations*

## 5.2 Loss of Synchronism/Out of Step Test

With the infinite bus taken as the reference, the power transfer of the machine can be expressed as Equation 5-1. If the excitation voltage is fixed, the curve to this equation is shown below:

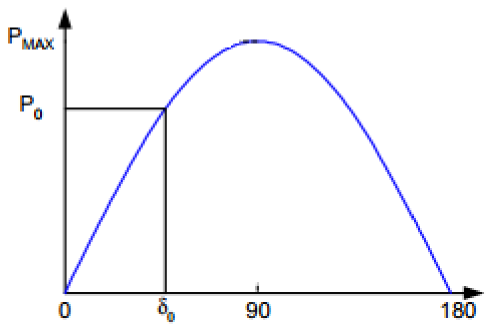

*Figure 5-4: Power Angle Curve*

In normal operation, $\delta$ stays below 90 degrees and all mechanical power, excluding losses, is transferred to the infinite bus electrically. The maximum power transferrable is reached when $\delta$ hits 90 degrees. After this point, electric power transferred decreases which increases the speed of the machine due to the swing equation. This critical point is known as the steady state stability limit. This synchronism limit should be tested by the encoder to see how well it will catch it.

To perform the loss of synchronism test, the Infinite Bus Connection Test was set up again with the machine producing zero power. The input power to the DC motor was then slowly increased which corresponded to a proportional increase in the input mechanical power of the synchronous generator. The increase had to be slow as a big step in input power to the generator could cause the machine to go over its transient stability limit due to the inertia of the machine. Figure 5-5 shows both the rotor angle and the terminal angle during the test while Figure 5-6 shows the torque angle, or difference between the two.

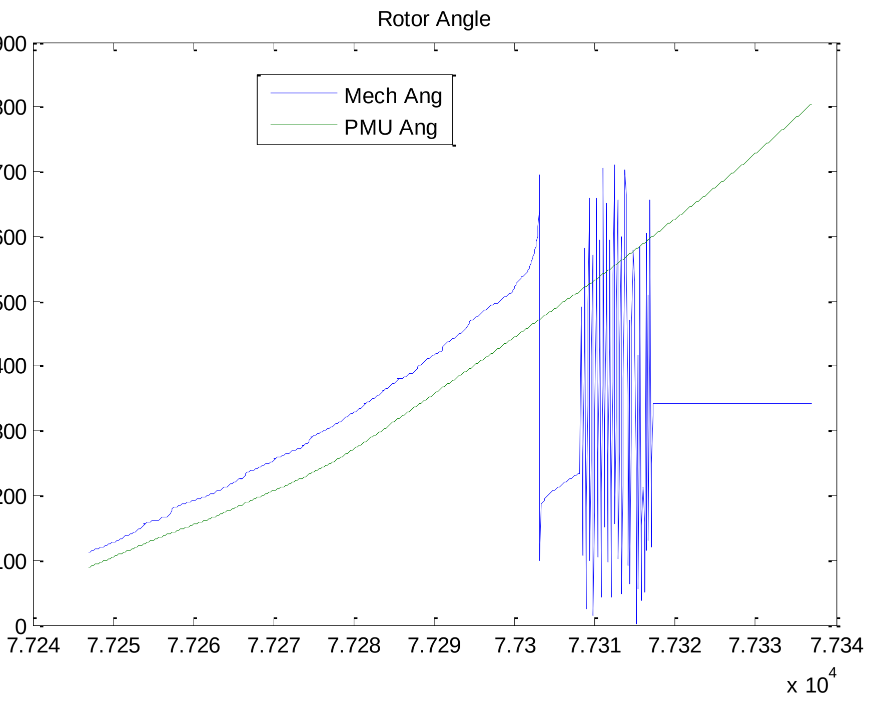

*Figure 5-5: Rotor Angle vs. Terminal Angle*

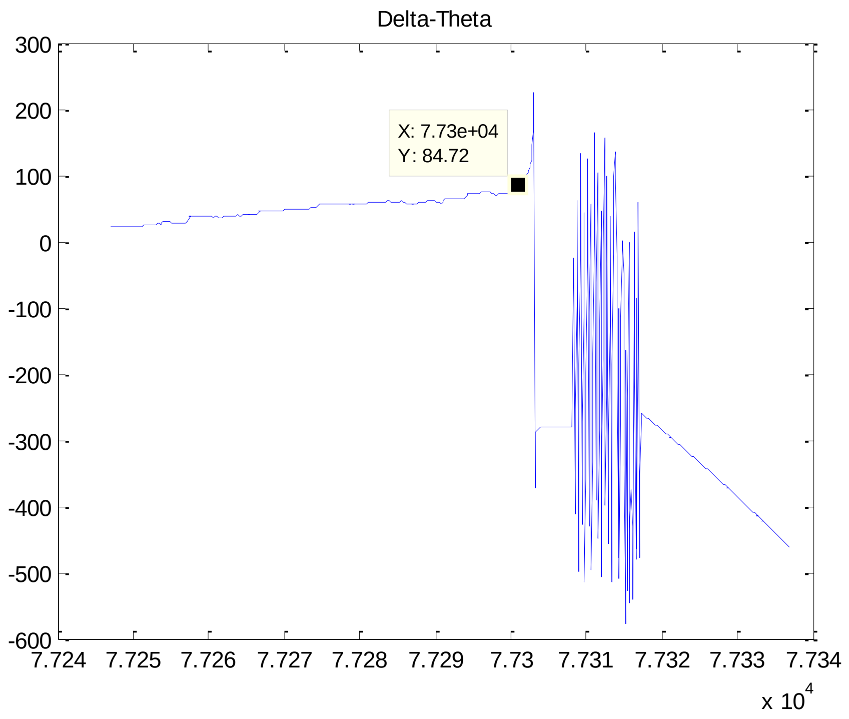

*Figure 5-6: Load Angle (Rotor Angle minus terminal angle)*

As can be seen from the graphs, the machine goes out of step at somewhere around 90 degrees. Getting the actual angle it lost stability at is extremely difficult. The best attempt was to use the value of the load angle graph where it was stable right before it lost stability. However, even this was extremely difficult. As seen from Equation 5-1, as the machine gets closer to its steady state stability limit, incremental changes in input power causes bigger changes in the load angle. Figure 5-7 illustrates this point. When the machine load angle is 85 degrees, the machine is producing 99.62% of its maximum capability. Any vibration in the DC motor or vibration in the DC source powering the DC motor that causes a power increase of 0.39% being delivered would cause the generator to lose synchronism.

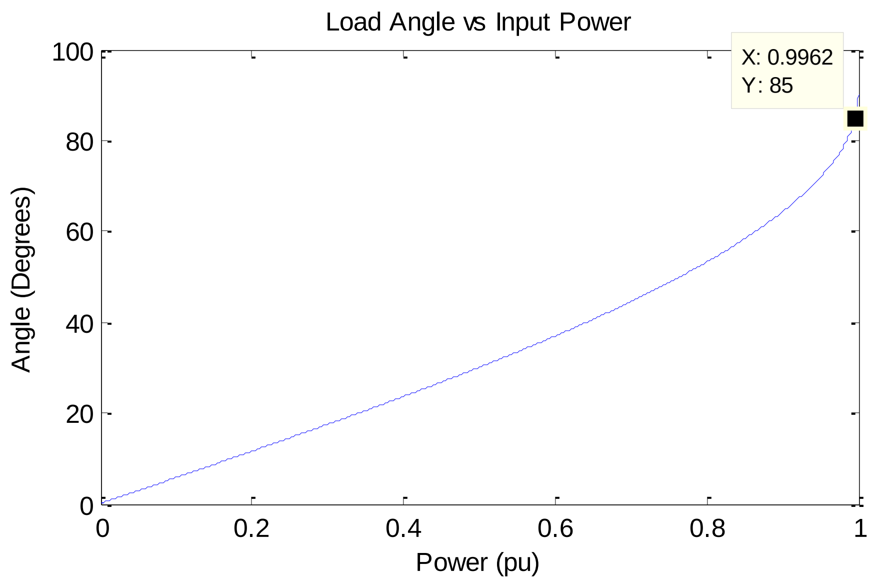

*Figure 5-7: Incremental Power Change*

The rotor angle measurement catches the loss of synchronism well. No further analysis has to be done to decipher whether the machine lost synchronism as with electrical quantities. A simple rate of change of angle will show that there is a spike in angular acceleration and velocity in the beginning with a step change in velocity after the machine completely goes out of synchronism. More results of the loss of synchronism can be seen in the Appendix III.

# 6 Power System Stabilizer

Power system Stabilizers (PSS) were created to help damp power system oscillations that occur in the system. These power system oscillations are small signal oscillations of approximately 0.3 to 3 Hz that occur between different generators due to their mechanical loops. The stabilizer uses an external signal from the power system to improve the dynamic performance of the system [8]. The PSS output is used as an input into a generator's excitation system as this is the most feasible place to control generator swings. There are many different input signals that a PSS can utilize including but not limited to mechanical shaft speed deviations, electrical frequency, and accelerating power with each signal having its own benefits and disadvantages [16].

## 6.1 Input Signals

### 6.1.1 Speed Deviation

The mechanical shaft speed deviations make the most sense to use since they are the actual quantities to be damped and are in direct phase with the damping torque required [8]. However, measuring the speed deviation of synchronous machines tends to amplify torsional modes of oscillation created by the shaft-turbine system. Torsional modes occur due to the machine shaft being slightly elastic and having some vibrations in them and are at much higher frequencies than conventional power swings. Torsional modes are excited on nearly all systems because their excitation systems have high gains at high frequencies that amplify these modes. If an excitation system excites one of these torsional modes, damage may be possible to the generator-turbine set. This is especially true at light loads where the damping of the mechanical torque is small [8]. Even if the excitation does not create damage to the shaft, the stabilizer output may still saturate due to these modes and become useless and possibly even saturate the voltage regulator could result in a loss of synchronism [8]. For this reason, and to limit the effect of noise, the gain of the PSS must be attenuated at high frequencies [17].

### 6.1.2 Electric Frequency

Electrical frequency tends to be more sensitive to inter-area modes of oscillations, or oscillations between separate plants and less sensitive to swings between individual units. However, the sensitivity of the frequency signal is highly dependent on the current power system [17], and may offset the action the controller is supposed to take on the electric torque of the machine [18]. As the system gets stronger, the frequency signal sensitivity drops. Frequency also tends to have sudden phase shifts following fast transients and large industrial load noise [19].

### 6.1.3 Accelerating Power

The accelerating power input uses electric power rather than speed because electric power is proportional to the rate of change of speed and is less subject to noise [16]. Electric power alone is not used because while mechanical power is generally constant, the governor can and will change it, causing an input signal that is not proportional to speed deviations. Accelerating power is defined as:

$$P_A = P_M - P_E \tag{6-1}$$

However, mechanical power is hard to measure [16]. For this reason, an electric power derivation is used to derive mechanical power using electric frequency. This signal is much less sensitive to torsional modes of oscillation.

### 6.1.4 Rotor Angle Measurement

With this synchronized delta angle measurement, a delta signal could also be used as an input to a power system stabilizer. Since the encoder is attached to the actual shaft, vibrational modes of oscillation will not be seen by the encoder. The rotational modes of oscillation will also be attenuated since the mechanical rotor angle position is the integral of the mechanical speed. Phase lag compensation can also be used on this signal as will be discussed in Section 6.2. This has the benefit of further decreasing the torsional modes of oscillation.

## 6.2 Power System Stabilizer Design

There are several methods to designing a power system stabilizer. The design goal should be to compensate for phase lag introduced between the exciter voltage input and output torque. This produces a torque that is in exact phase with speed oscillations and thus, a pure damping torque.

One common design approach is the lead-lag structure. Since the lag of the machine is what needs to be compensated, this is a nice simple structure to design. This follow the structure:

$$G_{PSS}(s) = K_{PSS} * \frac{1 + sT_1}{1 + sT_2} \tag{6-2}$$

where:

$$K_{PSS} = Power\ system\ Stabilizer\ Gain$$
$$T_1: lead\ compensator\ time\ constant$$
$$T_2: lag\ compensator\ time\ constant$$

If more than 90 degrees of compensation is needed, more than one of the lead-lag blocks can be used. Since the power system stabilizer is not meant to compensate for off-nominal frequencies, a washout filter has to be used. This has the form of:

$$H_{wash}(s) = \frac{sT_W}{1 + sT_W} \tag{6-3}$$

Generally, a time constant is chosen that is high enough to block out the near DC input values for off-nominal frequencies and oscillations while not messing with the inter-area oscillations of 0.3 Hz. A nice typical value is around 10 seconds. The PSS needs to be designed very carefully. It needs to compensate but not overcompensation. If the PSS over-compensates for the angle, then it will produce a negative synchronizing torque so the machine will not be connected to the system as strongly [8]. This can be seen in the figure below. If the PSS under-compensates (blue line), then an electric damping torque is produced that is in the direction of both angle and speed deviations. If the PSS compensates exactly (red line), it compensates only for speed deviations. However, if the PSS over-compensates (orange), an electric torque will be created that is in the direction of speed deviations but will be out of direction with angle deviations.

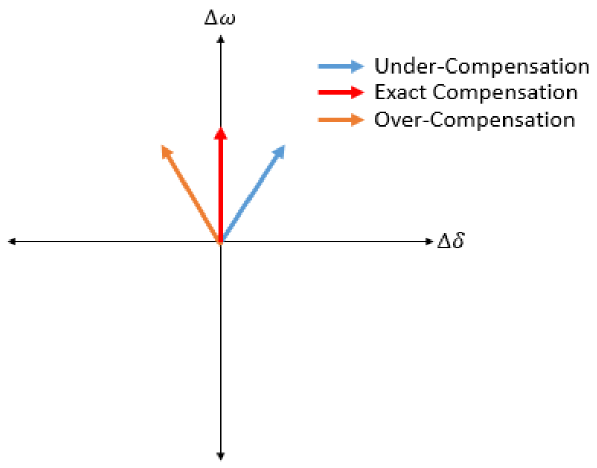

*Figure 6-1: PSS Compensations*

Figure 6-2 shows the model of how a PSS is connected. As can be seen, the GPSS needs to compensate for the phase in Gexc and the field circuit. Since the input in this paper is an angle deviation, extra lead compensation has to be used. Another option would be to invert the angle input signal. This would create a signal that is leading the required compensating angle and would thus require extra incorporated lagging angle. This has the benefit of decreasing torsional modes of oscillation and noise in the higher frequencies even further since lag compensation tends to act like integration. Figure 6-3 and 6-4 show lead and lag blocks, respectively. Creating an extra lead compensation in the angle signal measurements may offset the attenuation offset the angle signal has at torsional modes of oscillation. However, by inverting the input and adding a lag block to the PSS, the small signal oscillations are somewhat attenuated while the torsional frequency are greatly attenuated. Thus, a damping performance similar to speed stabilizers can be created with less gain at high frequencies.

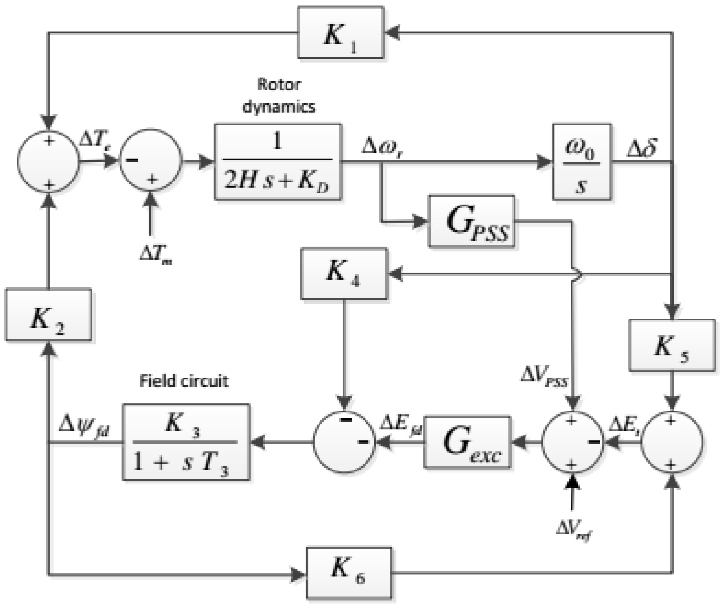

*Figure 6-2: Heffron-Phillips Model*

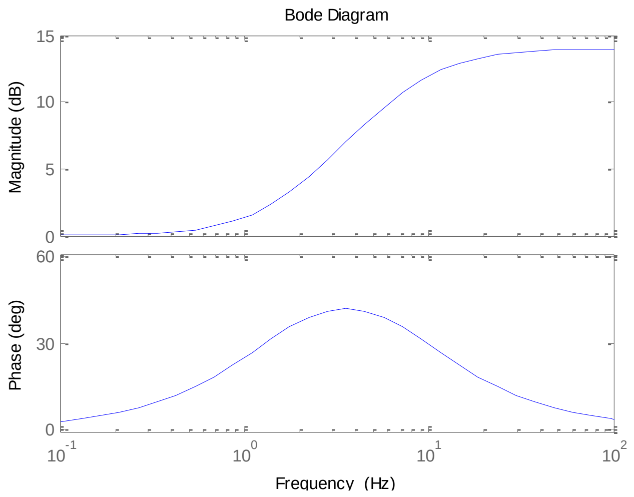

*Figure 6-3: Lead Block*

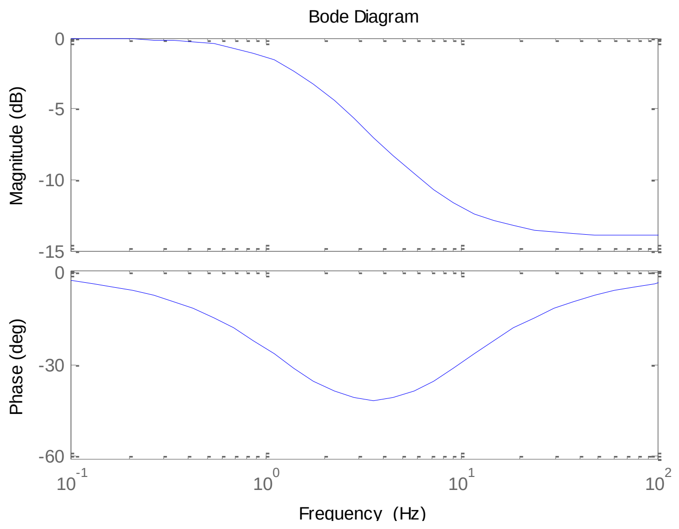

*Figure 6-4: Lag Block*

Since the remote rotor angle signal used in this work is a digital signal, it has odd harmonics of the fundamental, and its frequency content repeats after the Nyquist frequency defined as

$$f_{nyquist} = \frac{f_s}{2}. \tag{6-4}$$

These frequency images have to be filtered out with a low pass filter that does not affect the frequencies of interest to a PSS. Originally, the sampling frequency of the encoder angle was 10 Hz. This was because of the dynamics of interest in a machine have very low frequency content, and a small sampling rate helps with data collection. The simplified synchronous machine rotor speed equation is

$$M\frac{d\omega}{dt} = P_m - Pe \tag{6-5}$$

M can be thought of as a time constant for this differential equation. Typical values for M for a synchronous machine of 100 MVA is 2.5-5.3 [20]. As the machine gets bigger, that time constant grows due to the inertia of large generators. As can be seen from the time constant, typical frequencies of interest should be well below 5 Hz, the Nyquist frequency of this measurement system.

However, since the PSS's operating frequencies range from 0.3 to 3 Hz, and the Nyquist frequency is 5 Hz in this particular case, a low pass filter that could not affect the phase or magnitude of 3 Hz but completely cut off 5 Hz would have to be created. This would only be possible with an ideal low pass filter. For this reason, the sampling rate of the encoder should be increased from 10 Hz to 60 Hz. 60 Hz gives a Nyquist frequency of 30 Hz, which gives more than a decade for the filter to attenuate properly.

For this work's filter, a digital resonant second order low pass filter was designed and used with

$$H_{lowpass}(s) = \frac{1}{\frac{s^2}{\omega_n^2} + \frac{s}{Q\omega_n} + 1}. \qquad (6\text{-}6)$$

where

$$Q = 1.4$$
$$w_n = 2\pi * 10$$

$$H_{lowpass}(s) = \frac{1}{0.0002533s^2 + 0.01137s + 1}$$

Figure 6-5 shows the resultant bode plot of the low pass filter. There is some non-unity gain at 10 Hz and some phase lag addition at frequencies of interest, but they were minimal.

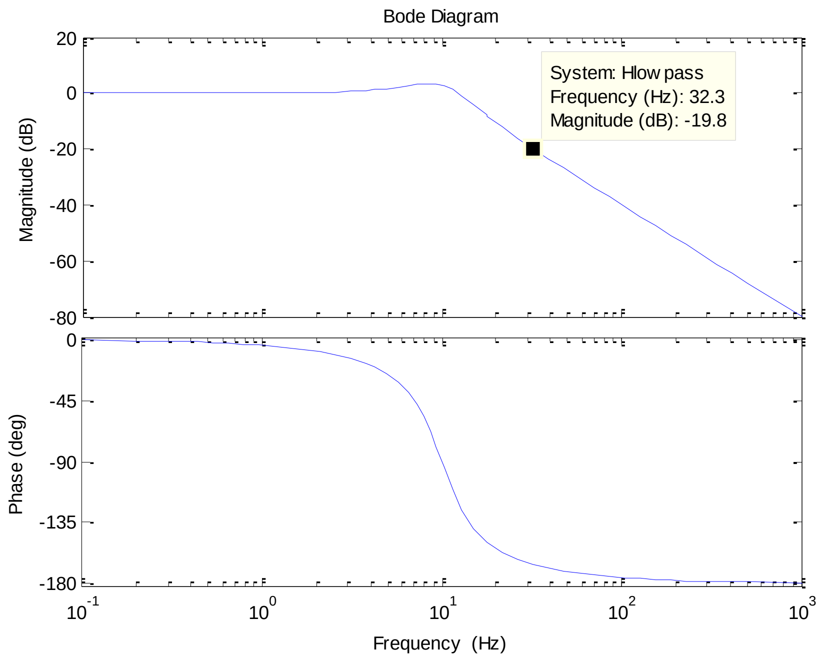

*Figure 6-5: 2nd Order Lowpass Filter*

## 6.3 System Analysis

To know what frequencies to compensate for, the system operating condition must be gotten. This can be done by linearizing the system around an operating point [21] and getting its open loop eigenvalues [22]. This is done through a state space representation that gives the end result:

$$\Delta\dot{x} = A\Delta x + B\Delta u \tag{6-7}$$

$$\Delta y = C\Delta x + D\Delta u. \tag{6-8}$$

where

$$x: System\ State$$
$$u: Inputs$$
$$y: Outputs$$

Therefore, the A matrix shows how the system naturally responds without any inputs. This is known as the open-loop matrix. B shows how the system responds to inputs, C shows how outputs responds to the current system state, and D shows how the outputs respond to inputs. As this is not the main concern of this paper, more information on state space representation and analysis can be gotten from [8], [19], [21], and [22].

### 6.3.1 Stability Analysis

Once the state space representation of the power system is gotten, its stability can be analyzed and determined. The analysis on the system's stability for this paper is a traditional eigenanalysis root-locus approach as those in [8], [21], and [23]. First, the eigenvalues of the system without any control action has to be found. These are the values that solve the equation

$$\underline{A\Phi} = \underline{\Lambda\Phi} \tag{6-9}$$

where A is the nxn open-loop matrix, $\Phi$ is the nxn modal column matrix containing the n right column eigenvectors, and $\Lambda$ is the nxn diagonal matrix containing the individual eigenvalues on its diagonals.

The stability of the system can be seen from these n eigenvalues. If any of the real parts of the eigenvalues are in the right hand plane, the system is unstable. This can be seen by looking at the time domain characteristics of the eigenvalues which are given by $e^{\lambda t}$, where $\lambda$ is the eigenvalue. If the real part of $\lambda$ is positive, as time goes on, the value will grow indefinitely towards infinity. The eigenvalues are in the form of

$$\lambda_i = \sigma_i \pm j\omega_{i.} \tag{6-10}$$

In this case, $\sigma$ is called the damping and $\omega$ is the frequency of oscillation. These values tell how the eigenvalue will oscillate around its steady state value, and how long it will take to reach that value. Experience shows that machine power system oscillations tend to be in the range of 0.3 to 3 Hz [22]. 0.3 to 0.7 Hz tend to be inter-area oscillations that happen between two separate coherent groups of generators in the system. 1-3 Hz tend to be local oscillations between individual units. For this project, the 4 machine two area system from [8] shown below was coded into Matlab's Simpower.

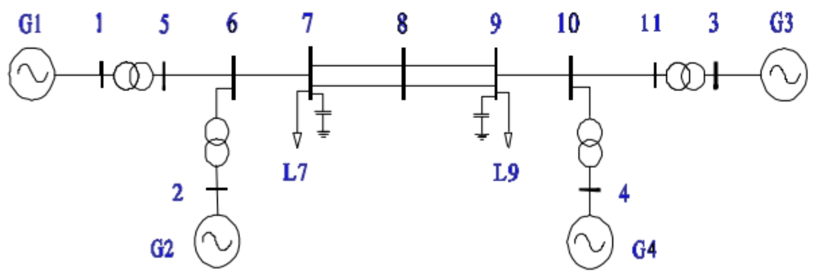

*Figure 6-6: System Model*

Linearizing the model and looking at the eigenvalues, the following machine oscillations were obtained:

*Table 6-1: Open Loop Damping and Frequencies (Hz)*

| Mode | Damping Factor | Frequency of Oscillation |
|---|---|---|
| **Local 1** | 0.1368 | 1.0924 |
| **Local 2** | 0.0967 | 1.1497 |
| **Inter-Area** | -0.0116 | 0.6345 |

As can be seen from the table above, there are two stable local oscillations of 1.09 and 1.15 Hz and one unstable inter-area oscillation of 0.635 Hz. Unfortunately, this does not tell which machines are participating in those oscillations, just that they are occurring. There is a rigid way obtaining the modes of oscillations associated with machine rotor angle deviations that will be discussed later.

## 6.3.2 Eigenvectors and Modal Analysis

Similarly to $\underline{\Phi}$, the n-row vector, $\Psi_i$, that solves

$$\Psi_i A = \lambda_i \Psi_i \qquad (6\text{-}11)$$

Is called the left-eigenvector. Once the right and left eigenvectors are obtained, the modal matrices can be defined:

$$\underline{\Phi} = [\Phi_1 \;\; \Phi_2 \;\; \dots \;\; \Phi_n] \tag{6-12}$$

$$\underline{\Psi} = [\Psi_1^{\mathrm{T}} \;\; \Psi_2^{\mathrm{T}} \;\; \dots \;\; \Psi_n^{\mathrm{T}}] \tag{6-13}$$

where:

$$\underline{\Psi}A = \Lambda\underline{\Psi} \tag{6-14}$$

$$\underline{A\Phi} = \underline{\Lambda\Phi} \tag{6-15}$$

Rearranging both equations to get their eigenvalue decompositions,

$$A = \Phi^{-1}\Lambda\Phi = \Psi^{-1}\Lambda\Psi \tag{6-16}$$

Solving for $\Lambda$,

$$(\Psi\Phi)\Lambda(\Phi^{-1}\Psi^{-1}) = \Lambda \tag{6-17}$$

Therefore, $\Phi\Psi = kI$. For convenience, k is set to unity so that

$$\Phi_i\Psi_i = 1 \tag{6-18}$$

and

$$\underline{\Psi} = \underline{\Phi}^{-1}. \tag{6-19}$$

As these two matrices are the inverse of one another, multiplying them together will create a dimensionless factor. This will become important when deciding which states are acting in the mode the most as seen in participation factors discussed in 6.3.5.

### 6.3.3 Decoupling System Modes

The current system is given by the state space representation as seen in section 5.3.

$$\dot{x} = Ax + Bu$$

$$y = Cx + Du$$

A is generally a full matrix so all states are coupled to one another. By defining a new state by

$$\Phi z = \Delta x \tag{6-20}$$

The new representation becomes

$$\dot{z} = \Lambda z + \Psi B \Delta u \qquad (6\text{-}21)$$

$$\Delta y = C\Phi z + D\Delta u \qquad (6\text{-}22)$$

Since $\Lambda$ is a diagonal matrix, the new states are decoupled, and it can be seen how certain inputs affect certain states as well as how certain outputs can observe states.

## 6.3.4 Mode Shapes

The response of a particular state variable in can be seen in that particular row of the mode of oscillation in $\Phi$. This is known as the mode shape of an oscillation mode. The combined information shows how each state variable is acting with regards to other state variables in that particular mode. The mode shapes of each eigenvalue can be checked to see if the rotor angle and rotor speed are oscillating against one another. The mode shapes of just the rotor angles of the three particular modes of oscillation have been plotted in Figures 6-7 to 6-9.

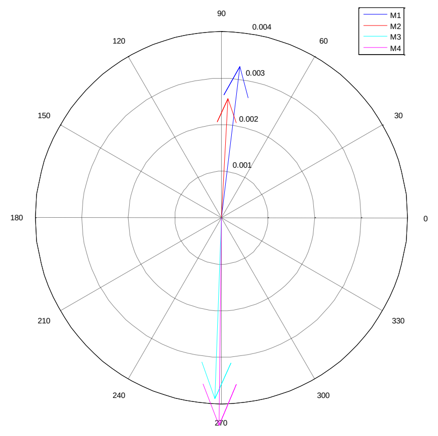

*Figure 6-7:Inter-Area Oscillation*

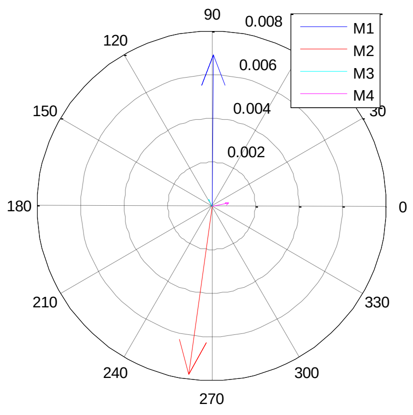

*Figure 6-8: Local Oscillation Area 1 – 1.09 Hz*

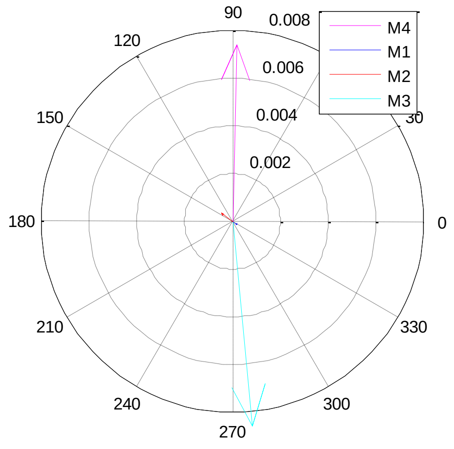

*Figure 6-9: Local Oscillation – Area 2: 1.15 Hz*

It is easy to see that in the 0.63 Hz oscillation that all the machine angles are participating. There are two groups of machines oscillating against one another (1 and 2 vs 3 and 4). This is therefore the inter-area oscillation. In the other two oscillatory modes, there is predominantly two generators swinging against one another (1 vs 2 and 3 vs 4). These are the two local oscillations.

### 6.3.5 Participation Factors

Mode shapes show how each state variable is participating in each mode, but it is difficult to compare the contribution of two separate state variables in the same mode as they are not the same quantity. Comparing speed deviations magnitudes to voltage deviation magnitudes does not make physical or mathematical sense. Originally proposed by [24], participation factors give a measure of how much each state is participating in a particular oscillatory mode. More specifically, the participation matrix is defined as

$$P = [p_1 \;\; p_2 \;\; \dots \;\; p_n] \tag{6-23}$$

with

$$p_i = \begin{bmatrix} p_{1i} \\ p_{2i} \\ \vdots \\ p_{ni} \end{bmatrix} = \begin{bmatrix} \Phi_{1i}\Psi_{i1} \\ \Phi_{2i}\Psi_{i2} \\ \vdots \\ \Phi_{ni}\Psi_{in} \end{bmatrix}. \tag{6-24}$$

$p_{ki} = \Phi_{ki}\Psi_{ik}$ is the participation factor of the kth state in the ith mode. Therefore, $p_i$ puts all the state variables in a measurable reference frame in the ith mode. This gives an indicator of which eigenvalues are due to the rotor angle oscillations. In this way, the different states can be compared in a mode of oscillation to see which are contributing the most to the oscillation. Participation factors are also useful in that not only can they show which of a single machine's states are participating in a mode of oscillation the most but also which machine's states are participating in the mode of oscillation. Therefore, participation factors can become the basis for optimal placement for PSS placement to increase the damping of that mode.

### 6.3.6 Controllability and Observability

Another thing to consider when trying to control a state is the controllability and observability of that state the placement of the PSS.

Considering the new decoupled state variable equations as seen in section 5.3.3,

$$\dot{z} = \Lambda z + \Psi B \Delta u$$
$$\Delta y = C\Phi z + D\Delta u$$

Define:

$$\overline{B} = \Psi B \tag{6-25}$$

$$\overline{C} = C\Phi \tag{6-26}$$

$\overline{B}$'s entries relate the inputs to the decouple states. Therefore, if the ith row of $\overline{B}$ is all zeroes, then the inputs do not affect the ith mode. That mode is said to be uncontrollable. Analogously, $\overline{C}$'s entries relate the decouple states to the output. If the ith column of $\overline{C}$ is zero, then that mode is said to be unobservable as the mode cannot be seen by the outputs. $\overline{B}$ and $\overline{C}$ are called the mode controllability and mode observability matrix, respectively [8].

### 6.3.7 Optimum PSS Placement

Ostojic used participation factors in [25] to develop a coupling factor using the basis of participation factor as mentioned earlier. The paper states that $\frac{\Phi_{jh}\Psi_{ih}}{M_i}$ will tell how much a perfect PSS applied on the ith machine affects the jth machine's motion during the hth oscillatory mode excitation due to generators coupling from energy exchange of oscillations between them. Therefore,

$$C_{ij_h}^2 = \frac{\Phi_{jh}\Psi_{ih}}{M_i} * \frac{\Phi_{ih}\Psi_{jh}}{M_j} = \frac{p_{ih}}{M_i} * \frac{p_{jh}}{M_j} \tag{6-27}$$

is the machine coupling factor between machine i and j in mode h. A total coupling factor between machines would then take into account all the modes of oscillation.

$$C_{ij}^2 = \sum_h C_{ij_h}^2 \tag{6-28}$$

In [26], a method of PSS optimal placement using residues was developed to optimize the controllability and/or the observability of a mode of oscillation. This work used the method of residues to obtain the proper placement. Using variables already defined in this work, the residue matrix is defined as

$$\overline{R} = \overline{B} * \overline{C}. \tag{6-29}$$

These residues are quantities that indicate how controllable and observable a state is. In other words, this shows how sensitive a mode of oscillation is to an input to a certain machine. The magnitudes of the residues for the three modes of oscillation of the test system are shown below.

*Table 6-2: Residue Magnitudes*

| | **Generator 1** | **Generator 2** | **Generator 3** | **Generator 4** |
|---|---|---|---|---|
| **Area 1** | 46.09 | 54.23 | 0.097 | 1.583 |
| **Area 2** | 1.51 | 0.25 | 44.86 | 45.96 |
| **Inter-Area** | 7.05 | 5.33 | 5.14 | 3.25 |

For the oscillation in area 1, the residue is highest on generator 1 and 2 while the residue is highest on generator 3 and 4 for the oscillation in area 2. A residue as low as seen in the local oscillations indicate that a stabilizer put on one of the generators in area 1 can neither see nor control the oscillations in area 2 and vice versa. Putting a stabilizer on generator 1 would be best for the inter-area oscillation and would benefit the local oscillation in area 1 significantly but would not do much for the oscillation in area 2.

## 6.4 Remote Feedback

In [19], Snyder showed that using a remote signal from the other area of oscillation in the stabilizer design increased either the controllability or observability of the inter-area oscillation as well as the controllability or observability in the area the remote signal came from. In the paper, he defines

$$\Delta u_{PSS} = \Delta P_a = Pa_{local} - Pa_{remote}. \tag{6-30}$$

Using this signal alone creates a much better controller for the power system oscillation. However, the paper assumes that the network transmission delay is zero as well as all the electric and mechanical powers being able to be measured in a synchronized manner. As there is no synchronized measurement of mechanical power, the assumption of derived mechanical power from electric frequency from [16] is assumed to have been used with PMU measurements. The assumption of no delay is validated on the basis of a dedicated communication network. However, this will not practically be the case, and more likely, the PMUs will be measuring over large areas that tend to go through many switches and Ethernet lines with delay. [19] states that a delay larger than 200 ms significantly reduces the performance of the power system stabilizer. This tends to be on the order of magnitude of communication delays with switches.

Using the synchronized rotor angle measurement, this paper proposes the new input as

$$\Delta u_{PSS} = \Delta\delta = \delta_{local} - \delta_{remote}. \qquad (6\text{-}31)$$

The PSS using this signal was tuned as in [8] with the extra lead compensation required for the phase. Analysis of the eigenvalues using this method and using the local signal method are compared in the tables and figures below. Overall, adding the remote signal to the PSS has increased the stability of the system.

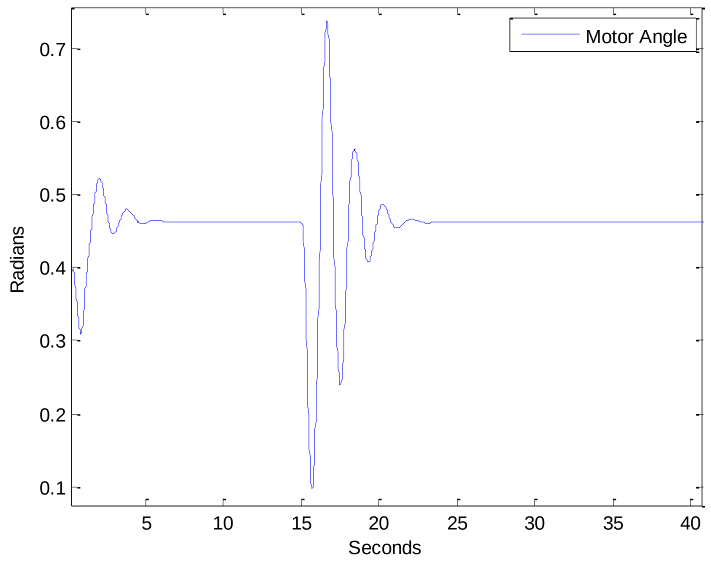

*Figure 6-10: Remote Signal Feedback*

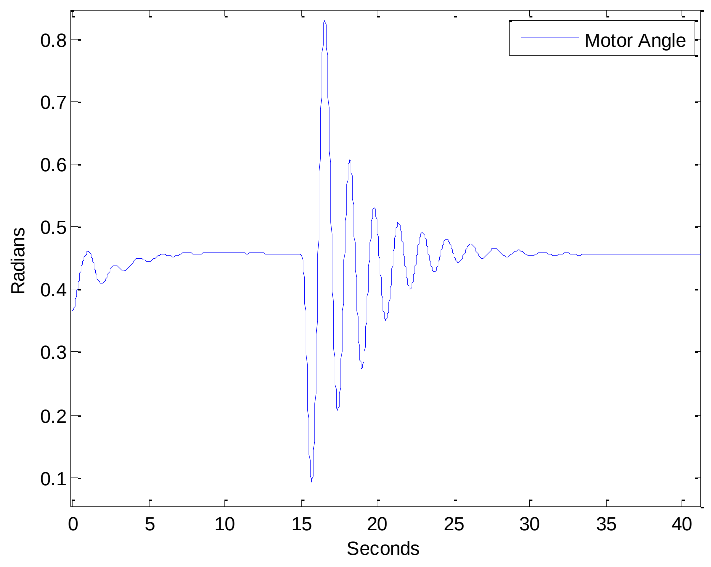

*Figure 6-11: Local Signal*

*Table 6-3: Damping and Frequencies with Only Local Signal*

| **Mode** | **Damping Factor** | **Frequency of Oscillation** |
|---|---|---|
| **Local 1** | 0.614 | 0.92 |
| **Local 2** | 0.0967 | 1.1497 |
| **Inter-Area** | 0.0908 | 0.6463 |

*Table 6-4: Eigenvalue damping and Frequencies with Remote Feedback*

| **Mode** | **Damping Factor** | **Frequency of Oscillation** |
|---|---|---|
| **Local 1** | 0.618 | 0.87 |
| **Local 2** | 0.0925 | 1.15 |
| **Inter-Area** | 0.219 | 0.694 |

The local area oscillation in one has been greatly damped while the oscillation in area 2 remains relatively undisturbed. The damping on the inter-area oscillation has increased by a factor of 2.4 when using the remote signal. The benefit of using this signal over the accelerating power signal proposed in [19] is that the rotor angle is very slow moving. It does not see sudden phase shifts the accelerating power signal sees during transients. As it's a slow moving quantity, it can be predicted with more sufficient accuracy to help accommodate for delay or loss of information from the remote feedback signal.

## 6.5 Remote Signal Prediction

To predict the remote signal rotor angle, a polynomial prediction method was chosen. The model can be expressed as:

$$\underline{\delta}(t) = a_0 + a_1 t + a_2 t^2 + \cdots + a_n t^n \tag{6-32}$$

$$A = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \tag{6-33}$$

where

$\underline{\delta}$: *predicted value of rotor angle*

$t$: *time*

$A$: *polynomial parameter vector*

The A parameter vector can be gotten with linear regression using the Least-Squares method. More information on the least-squares method can be gotten from [28]. The parameter vector thus becomes:

$$A = \left(H^T(N)H(N)\right)^{-1} H^T(N) * Y(N). \tag{6-34}$$

Letting $\Delta t$ be the sampling period, the observation vector $Y(N)$ and $H(N)$ becomes

$$Y(N) = \begin{bmatrix} \delta(0) \\ \delta(\Delta t) \\ \vdots \\ \delta(N\Delta t) \end{bmatrix} \tag{6-35}$$

$$H(N) = \begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & \Delta t & (\Delta \mathrm{t})^2 & \dots & (\Delta t)^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & N\Delta t & (N\Delta t)^2 & \dots & (N\Delta t)^n \end{bmatrix} \tag{6-36}$$

Once A is gotten, the i^th future value of the prediction can be gotten from

$$\underline{\delta}(k\Delta t) = a_0 + a_1 k\Delta t + \cdots + a_n (k\Delta t)^n \tag{6-37}$$

where:

$$k = N + i.$$

After some testing, it was found that a 2nd order polynomial estimation was the most accurate along with 5 observations. [29] has found that adding more observations after this point does not help the prediction accuracy. This method of predicting rotor angle could be accurate for up to a 0.25 to 0.3 second prediction time period using the rolling prediction method of updating the parameter vector as new measurements came in. To simulate the digital feedback signal, a zero order sample and hold was used on the remote delta measurement in the simulation and a delay of $z^{-15}$ with a sampling period of 16.66 ms was used to simulate a communication delay of 0.25 seconds. Figure 6-10 shows the result after a 20% increase pulse on the reference voltages to the machines in Area 1. This was to extremely excite the modes of oscillation in the area. Figure 6-11 shows a fault occurrence at 20 seconds with a 10 cycle breaker clearing time. Note that while the predictor was predicting these, no PSS was implemented. The purpose of this was to test the predictor under the most extreme cases.

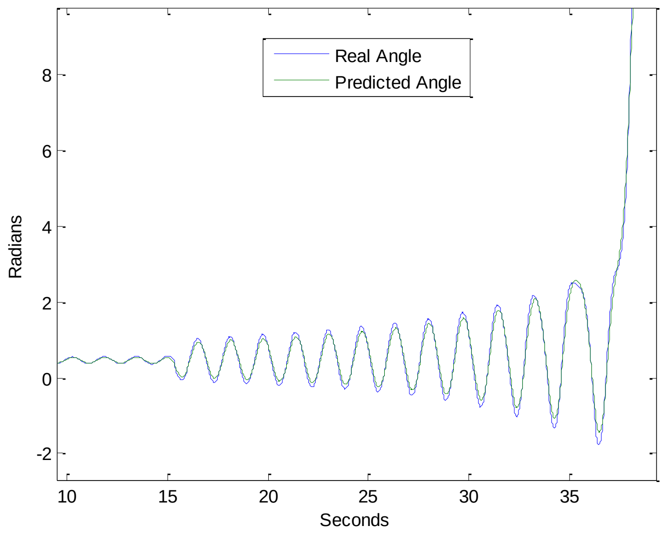

*Figure 6-12: Predicted Angle 20% Step Change on Area 1 Excitation*

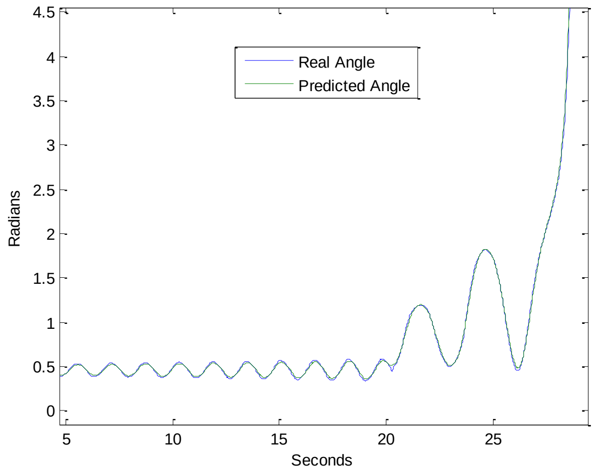

*Figure 6-13: Predicted Angle Fault with 10 cycle Breaker Clearing*

The machines quickly lose synchronism, but the predictor remains accurate regardless. As can be seen, even after severe events like a fault, the rotor angle moves relatively slowly and can thus be predicted to help compensate for communication delay between the remote and local machine angle measurements. This prediction was then tested on the accelerating power signal. Again, no mechanical measurement was made. Instead mechanical power was inferred from the electric frequency. Figure 6-12 shows one sample ahead (16.66 ms) prediction of the accelerating power of the machine under the same conditions as Figure 6-11. It does well under non-transient, quasi-steady state conditions. However, the transient conditions perturb a phase change in the accelerating power that cannot be predicted. This creates an extreme overshoot that could make a machine close to its stability limit lose stability.

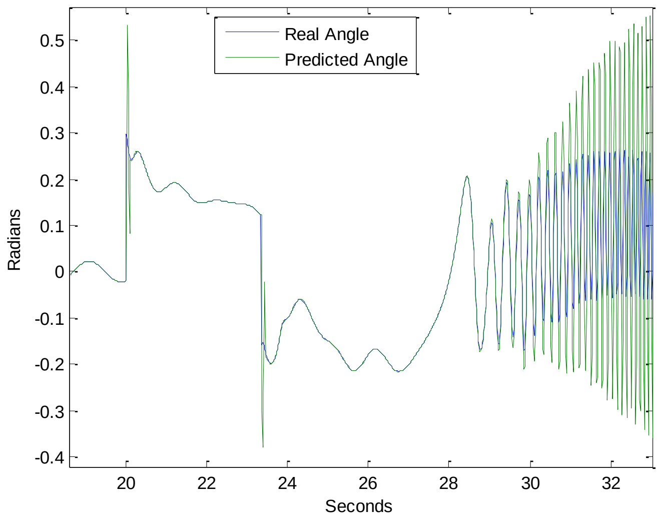

*Figure 6-14: Accelerating power with Fault*

When the machine starts losing synchronism and starts oscillating extremely, the accelerating power prediction loses all accuracy. This prediction cannot be used safely for control even with one sample ahead.

The same situation as the remote feedback in section 6.4 was simulated with the remote angle feedback signal being predicted by 0.25 seconds ahead. However, the time domain simulations of both situations have been plotted in Figure 6-13. It can be seen that the results are almost completely identical. The beginning is decently far off as the predictor needs to fill its window with measurements.

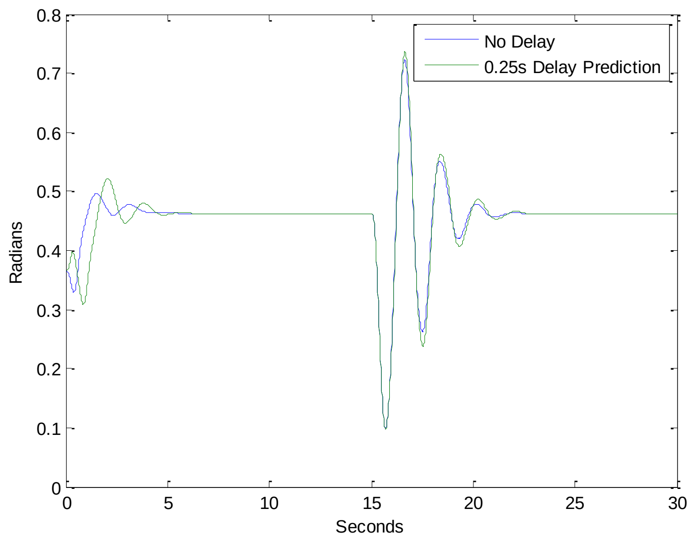

*Figure 6-15: Predicted Remote Signal Control*

# 7 Other Applications

## 7.1 Generator Protection

As mentioned in section 5.1, the rotor angle measurement is sensitive to torsional modes of oscillation or sub-synchronous resonance (SSR). While this isn't wanted for an input of a PSS, it is helpful to be able to capture it with some measurement tool. SSR can cause electrical instability and turbine-generator shaft failure. The torsional interaction may occur when the natural frequency of the mechanical system is near the resonant frequency of the electrical system [30].

As mentioned earlier, a PMU will not be able to capture those modes of oscillation due to the electrical frequency damping out the torsional mode frequencies due to standard requirements [31]. On top of the electric frequency naturally attenuating the torsional modes, PMUs also have filtering characteristics that will filter out SSR. Even the wide frequency response of PMUs has attenuations at the SSR frequencies [30]. However, a synchronized angle measurement would not need this type of filter response. The only requirement for it would be filtering for noise and anti-aliasing.

## 7.2 Machine parameter estimation/validation

The angle measurement can help to do machine parameter estimation and validation. Using voltage and current measurements at the terminals of the machine along with the internal rotor angle measurement, a possible real-time machine parameter estimation can be made. A possible concept for transient reactance and internal voltage is shown below that can be used in transient stability control [31].

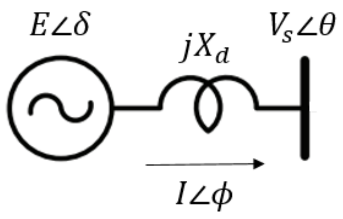

*Figure 7-1: Classical Model*

The equation for 7-2 is given by:

$$Ee^{j\delta} = Ve^{j\theta} + (jX_d)Ie^{j\phi}. \tag{7-1}$$

Splitting (7-1) into real and imaginary and equating the two, the following is obtained.

$$Ecos(\delta) = Vcos(\theta) - X_d Isin(\phi) \tag{7-2}$$

$$Esin(\delta) = Vsin(\theta) + X_d Icos(\phi) \tag{7-3}$$

Referencing both equations to $\delta$

$$E = Vcos(\theta - \delta) - X_d Isin(\phi - \delta) \tag{7-4}$$

$$X_d = -\frac{Vsin(\theta)}{Icos(\phi)} \tag{7-5}$$

With these values now gotten, studies in transient stability, including transient energy functions or Equal Area Criterion, can be done much more accurately as the rotor angle is not estimated by the generator's electric terminal angle.

In other more detailed parameter estimation, such as [32], the following parameter model is developed. For full derivation of this model, see [32].

$$\begin{bmatrix} v_o \\ v_d \\ v_q \\ -v_F \\ v_D \end{bmatrix} = \begin{bmatrix} r + 3r_n & 0 & 0 & 0 & 0 \\ 0 & r & \omega L_q & 0 & 0 \\ 0 & -\omega L_D & r & -\omega k M_F & -\omega k M_D \\ 0 & 0 & 0 & r_F & 0 \\ 0 & 0 & 0 & 0 & r_D \end{bmatrix} \begin{bmatrix} i_0 \\ i_d \\ i_q \\ i_F \\ i_D \end{bmatrix} - \begin{bmatrix} L_0 + 3L_n & 0 & 0 & 0 & 0 \\ 0 & L_d & 0 & kM_F & kM_D \\ 0 & 0 & L_q & 0 & 0 \\ 0 & kM_F & 0 & L_F & M_R \\ 0 & kM_D & 0 & M_R & L_D \end{bmatrix} \begin{bmatrix} \dot{i_0} \\ \dot{i_d} \\ \dot{i_q} \\ \dot{i_F} \\ \dot{i_D} \end{bmatrix} \tag{7-6}$$

where

$$k = \sqrt{\frac{2}{3}}$$

$$L_d = L_S + M_S + \frac{3}{2}L_m \tag{7-7}$$

$$L_q = L_S + M_S - \frac{3}{2}L_m \tag{7-8}$$

$$L_0 = L_S - 2M_s \tag{7-9}$$

(7-6) is then rearranged in the linear state space representation as given in section 5.3. However, it is assumed that $\omega$ is almost constant, therefore (7-6) can be considered a linear equation. This would not be the case in instances of severe dynamics. With the synchronized delta measurement, $\omega$ can be derived from the encoder measurements and used as a measurement. This would make (6-6) a truly linear equation from step to step with the parameter matrices becoming time variant.

In [33], the authors assume that the rotor angle is a measurement and achieve decent results incorporating even saturation. [34] uses PMU measurements to try to estimate the transient reactance and voltage of the machine using a process similar as described above. However, since using only PMU measurements makes it an underdetermined system, the authors had to make the assumption that the induced internal voltage is constant and as such, the transient reactance can be related to the terminal voltages by use of a least squares estimator.

## 7.3 Center of Inertia for Angular Stability

Center of inertia, or center of angles, is a metric that can be used to determine if multiple systems are held together or are drifting apart [35], [36]. It is defined as:

$$\delta_{COA}^{i} = \frac{\sum_{i=1}^{N} \delta_i H_i}{\sum_{i=1}^{N} H_i}. \tag{7-10}$$

This gives a measure of where the angle for a group of generators is acting. Thus a center of angles could be gotten for a group of coherent group of generators, and the center of angle calculated for all the generators could be calculated as a reference with

$$\delta_{Ref} = \frac{\sum_{i=1}^{N} \delta_{COA}^{i} H_{COA}^{i}}{\sum_{i=1}^{N} H_{COA}^{i}}. \qquad (7\text{-}11)$$

When the area representative angle, $\delta_{COA}^{i}$, moves away from the reference angle continuously, it could be heuristically determined that that area is moving away from the rest of the system [36]. However, the problem arises that neither the inertia of the machine nor the internal angle can be measured directly. To get past this, injected power is used as an index of machine size instead of the inertia of the generator, and the high side bus voltage angle is used instead of the actual internal angle of the generator.

This representative electrical angle may not, however, give a decent approximation of the internal angle if the system is heavily loaded as the internal angle will be much larger. It may also not be very representative when multiple generators are paralleled onto the same bus.

# 8 Conclusion and Future Work

## 8.1 Conclusion

In this document, the author has presented a method for measuring and synchronizing the rotor angle of a machine. It is a simple measurement and requires minimal filtering for anti-aliasing and noise on the measurement. The angle has a constant offset error that needs to be fixed in future work. It accurately follows the steady state as well as even the most severe transient dynamics such as pole slipping. Therefore, it is possible to measure accurate machine dynamics and send it synchronized for stability analysis or control.

The paper then follows the measurement up with a possible wide-area application: power system stabilizers. To use the rotor angle as an input to the stabilizer, the classic tuning methods need to have an extra 90 degrees of lead compensation, or the input can be inverted with lag compensation incorporated. In the power system stabilizer, the synchronized rotor angle measurement is used to increase the observability and controllability of an inter-area mode of oscillation with respect to a power system stabilizer installed on one machine. It does this by making the new stabilizer input the difference between the local and remote angle measurements. If the two areas are oscillating against one another, this will amplify that oscillation and make the controller more efficient in damping the oscillation. It was shown in the four machine system in this work that using the wide-area signal increases the stability of the system.

Afterwards, the communication delay of this signal was taken into consideration. The rotor angle is a slow, smooth and continuous moving function due to the inertia of the machine. Therefore, the angle can be reliably predicted for a small period of time. It was found that a 2$^{nd}$ order polynomial worked well for predicting the rotor angle up to 0.25 seconds. This prediction was then used in the four machine system to show that the unstable system could be made stable again with this prediction.

## 8.2 Future Work

Future work of this paper includes improvement of microprocessor clock accuracy, improvement of reference setup for the encoder initial value, a test on a paralleled machine setup with local dynamic tests, and bad data detection of the rotor angle.

The first task can be done in one of two ways. The first and easiest method would be to get a microprocessor with a much higher accuracy oscillator. This would solve all the issues but would require buying a new microprocessor with a new hardware configuration and capabilities. The second method would be to replace the oscillator on the current board with a higher accuracy temperature controlled oscillator. As the current oscillator, with the PTP software modifications is around 10 ppm, an oscillator better than that would have to be considered. Temperature oscillators tend to have a maximum accuracy of around 1.5 ppm and get very expensive. A drawback of using this method would be that extreme knowledge of oscillators would have to be known to put the new oscillator in place. With oscillators at that frequency, the capacitance and inductance of the wires around them affect their output.

The second task requires an improvement of the referencing technique. This paper's encoder referencing technique left a couple of degrees of mechanical play on the shaft of the generator. This may not be a problem on larger generators with large inertias but one cannot know for sure. Another way of referencing would be to strictly line up the d-axis with the magnetic field of phase A or 90 degrees lagging the middle of the A phase coils.

The parallel test would require a second generator and sizable load to be placed at or close to the terminals of the generator. The purpose of this experiment would be to see the dynamics that the encoder reading can see between the two generators that the Phasor Measurement Unit filters out. It was already shown in this paper that the encoder could see oscillations that the PMU could not, but a further test would be to have two generators in the lab oscillating against one another with digital controllers adding to the exciter inputs to help damp the oscillations.

Bad data detection has been a hot topic in PMU research, and those same techniques could be applied to the rotor angle. The detection for the rotor angle would be much simpler than in PMU's since the rotor is such a slow moving entity.

# References

[1] V. Kinitsky, "Measurement of Rotor Diplacement Angle of Synchronous Machines," *Power Apparatus and Systems, Part III. Transactions of the American Institute of Electrical Engineers,* vol. 77, no. 3, pp. 349-352, 1958.

[2] D. Lewis, "Measurement of Rotor Load Angle of Synchronous Machines," *Electronics Letters,* vol. 5, no. 6, pp. 113-114, 1969.

[3] D. Huber and O. Malik, "A Digital Device to Measure Angular Speed and Torque Angle," *IEEE Transactions on Industrial Electronics and Control Instrumentation,* Vols. IECI-22, no. 2, pp. 186-188, 1975.

[4] S. I. Ahson and M. H. Ali, "A Microprocessor-Based Scheme for Torque-Angle and Speed Measurement," *IEEE Transactions on Industrial Electronics,* Vols. IE-34, no. 2, pp. 135-138, 1987.

[5] M. Omoigui, "Precision measurement of Rotor Angle of a Microsynchronous Alternator," *IEEE Transactions on Instrumentation and Measurement,* vol. 39, no. 6, pp. 1045-1047, 2002.

[6] F. de Mello, "Measurement of Synchronous Machine Rotor Angle from Analysis of Zero Sequence Harmonic Components of Machine Terminal Voltage," *IEEE Transactions on Power Delivery,* vol. 9, no. 4, pp. 1770-1777, 2002.

[7] D. C. Mazur, "Synchronized Rotor Angle Measurement of Synchronous Machines," Blacksburg, VA, 2012.

[8] P. S. R. Committee, "C37.118-2005 IEEE Standard for Synchrophasors for Power Systems," IEEE, New York, 2005.

[9] Sick, "AFS60/AFM60 SSI Absolute Encoders," 2012.

[10] EPSON, *Programmable High-Frequency Crystal Oscillator SG-8002JF series,* 2001.

[11] Global Top, "Gms-u1LP GPS Module Data Sheet," GLobalTop Technology Inc., Tainan, Taiwan, 2010.

[12] Sick/Stegmann, "Synchronous Serial Interface for Absolute Encoders," Stegmann, Germany, 2004.

[13] MatrixLab, "MatrixLab Examples," SBI, 2009. [Online]. Available: http://www.matrixlab-examples.com/gray-code.html. [Accessed 3 November 2014].

[14] T. Schmidt, "CRC Generating and Checking," Microchip Technology Inc., 2000.

[15] P. Kundur, Power System Stability and Control, McGraw-Hill,Inc, 1994.

[16] F. De Mello, L. Hannett and J. Undrill, "Practical Approaches to Supplementary Stabilizing from Accelerating Power," *Power Apparatus and Systems, IEEE Transactions on,* Vols. PAS-97, no. 5, pp. 1515-15222, Sept 1978.

[17] E. Larsen and D. Swann, "Applying Power System Stabilizers Part 1: General Concepts," *IEEE Transactions of Power Apparatus and Systems,* Vols. PAS-100, no. No. 6, 1981.

[18] E. Larsen and D. Swann, "Applying Power System Stabilizers Part 2: Performance Objectives and Tuning Concepts," *IEEE Transactions of Power Apparatus and Systems,* vol. PAS_100, no. No. 6, 1981.

[19] A. Snyder, "Inter-Area Oscillation Damping with Power System Stabilizers and Synchronized Phasor Measurements," VT, Blacksburg, 1997.

[20] C. L. Wadhwas, Electrical Power Systems, Daryaganj: New Age International, 2005.

[21] B. Friedland, Control System Design: An Introduction to State-Space Methods, New York: Dover, 2005.

[22] P. W. Sauer and M. A. Pai, Power System Dynamics and Stability, Stipes Publishing LLC, 2007.

[23] B. Kuo, Automatic Control Systems, New Jersey: Prentice-Hall, 1987.

[24] G. Verghese, I. Perez-Arriaga and F. Schweppe, "Selective Modal Analysis with Application to Electric Power Systems, Part I: Heuristic Introduction, Part II: The Dynamic Stability Problem," *IEEE Transactions,* Vols. PAS-101, no. 9, pp. 3117-3134, 1982.

[25] D. Ostojic, "Identification of Optimum Site for Power System Stabiliser Applications," *Generation, Transmission and Distribution,* vol. 135, no. 5, pp. 416-419, 2002.

[26] D. Ostojic, "Stabilization of Multimodal Electromechanical Oscillations by Coordinated Application of Power System Stabilizers," *IEEE Transactions on Power Systems,* vol. 6, no. 4, pp. 1439-1445, 2002.

[27] C. R. Rao, H. Toutenburg, Shalabh, C. Heumann and M. Schomaker, Linear Models and Generalizations: Least Squares and Alternatives, Ney York: Springer Berline Heidelberg, 2007.

[28] K. Men, P. Xu, J. Zhao, X. Wu and C. Hong, "Comparison of Methods for the Perturbed Trajectory Prediction Based on Wide Area Measurements," in *Power Engineering and Automation Conference*, Wuhan, 2011.

[29] M. Elfayoumy and C. Moran, "A Comprehensive Approach for Subsynchronous Resonance Screening Analysis Using Frequency Scanning Tecqnique," in *Power Tech Conference Proceedings*, Bologna, 2003.

[30] G. Zweigle, D. Finney and R. Moxley, "Adding Shaft Angle Measurement to Generator Protection and Monitoring," in *Annual Conference for Protective Relay Engineers*, College Station, Tx, 2013.

[31] E. Kyriakides and G. Heydt, "Synchronous Machine Parameter Estimation Using a Visual Platform," in *Power Engineering Society Summer Meeting*, 2001.

[32] H. Tsai, A. Keyhani, J. Demcko and R. Farmer, "On-line synchronous Machine Parameter Estimation from Small Disturbance Operating Data," *IEEE Transactions on Energy Conversion,* vol. 10, no. 1, pp. 25-36, 2002.

[33] Y. Wehbe, L. Fan and Z. Miao, "Least Squares Based Estimation of Synchronous Generator States and Parameters with Phasor Measurement Units," in *North American Power Symposium (NAPS)*, Champaign, IL, 2012.

[34] D. Hu and V. Venkatasubramanian, "New Wide-Area Algorithms for Detection and Mitigation of Angle Instability using Synchrophasors," in *Power Engineering Society General Meeting*, Tampa, FL, 2007.

[35] C. Taylor, D. C. Erickson, K. Martin and R. Wilson, "WACS-Wide-Area Stability and Voltage Control System: R&D and Online Demonstration," *Proceedings of the IEEE,* vol. 93, no. 5, pp. 892-906, 2005.

[36] W. Watson and G. Manchur, "Experience with Supplementary Damping Signals for Generator Static Excitation Systems," *Power Apparatus and Systems, IEEE Transactions on,* Vols. PAS-92, no. 1, pp. 199-203, Jan 1973.

[37] RUGGEDCOM, Latency on a Switched Ethernet Network, Woodbridge, 2008.

## Appendix A – Experimental Results

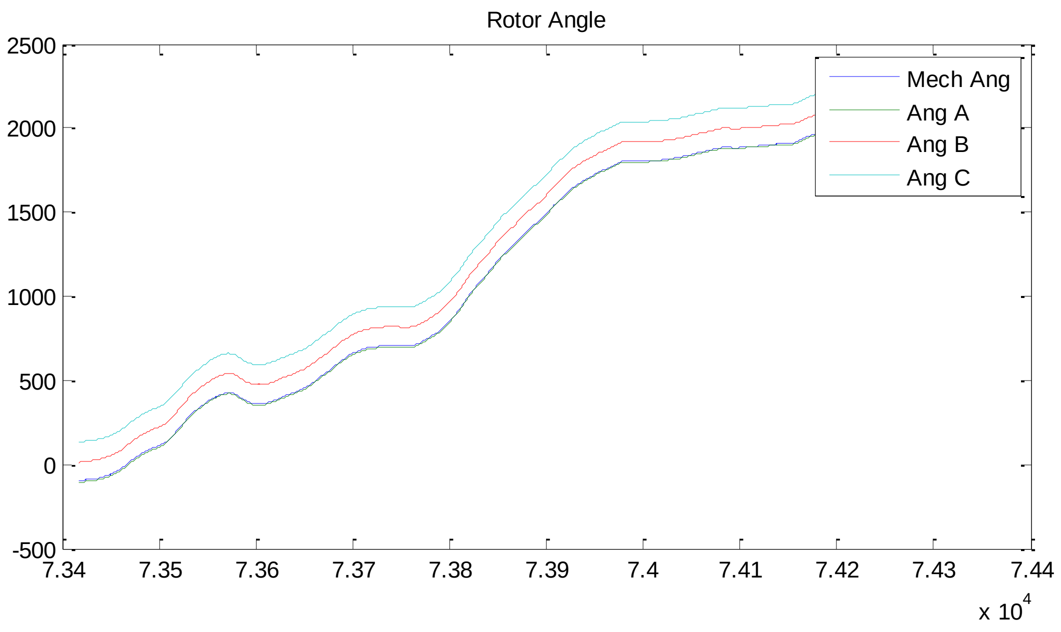

*Figure A-1: Rotor Angle and Three Phases*

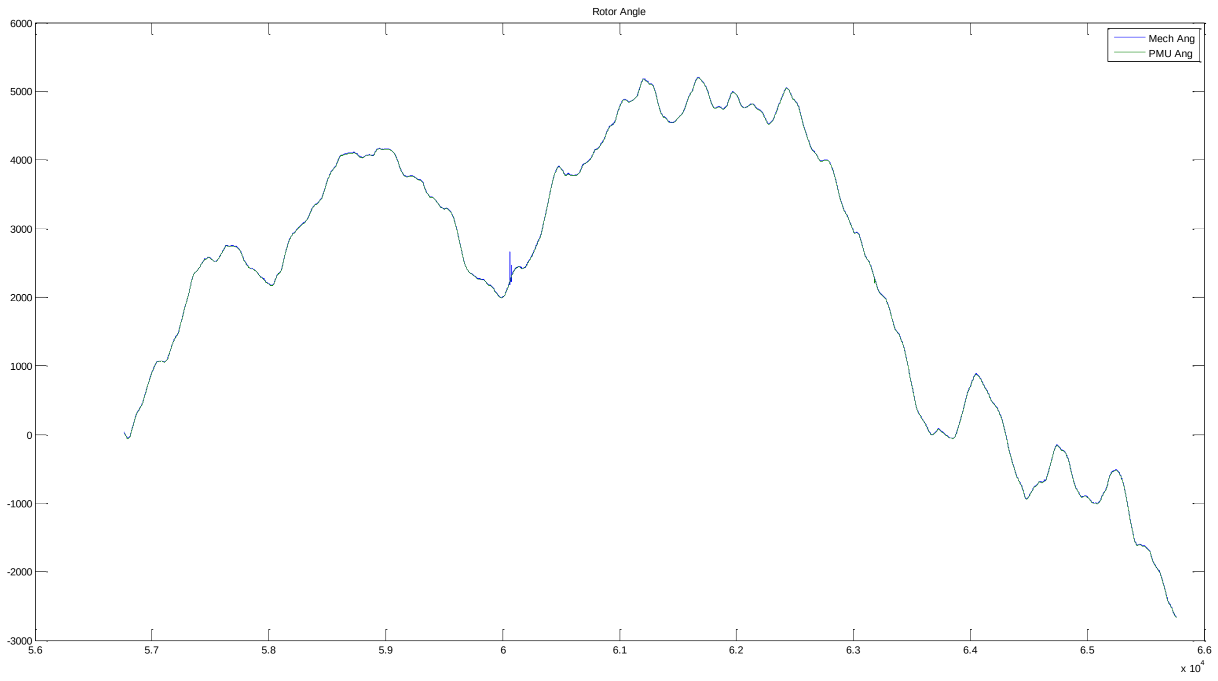

*Figure A-2: 2 Hour Test*

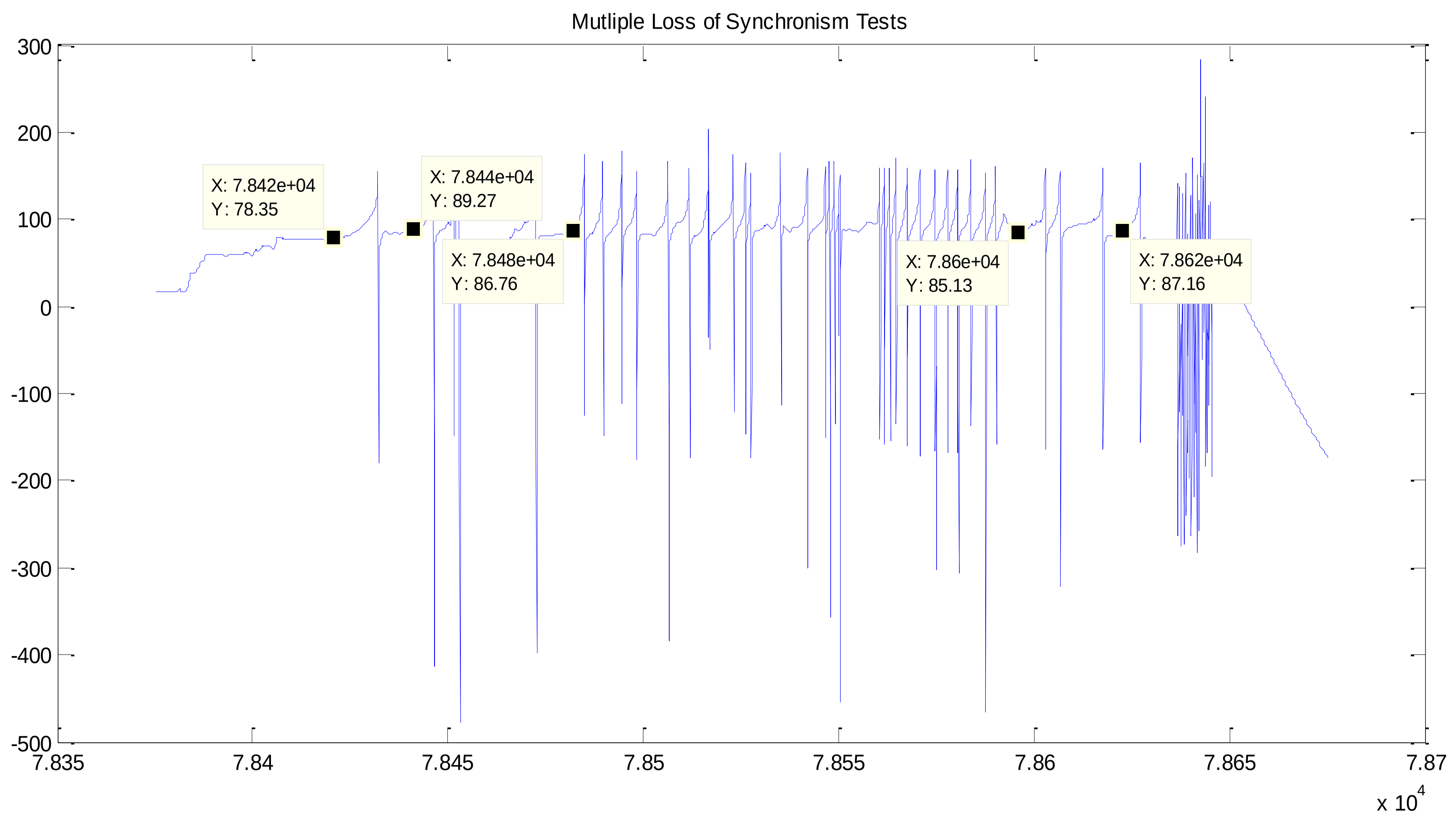

*Figure A-3: Loss of Synchronism Multiple Times*

## Appendix B – Real Time PMU and Encoder Plotting Code

```
% function Serial_Reader_plotter()
%% UDP Set up
if(exist('u','var'))
    flushinput(u);
    fclose(u);
    delete(u);
    clear u;
end
if(exist('s','var'))
    flushinput(s);
    fclose(s);
    delete(s);
    clear s;
end
close all;
clear all;
% dbstop in Serial_Reader_plotter at 166
%% Serial Initialization
SerialPort='com3'; %serial port
Rate = 50000;

Data_bits=8;
%%Set up the serial port object
s =
serial(SerialPort,'BaudRate',Rate,'DataBits',Data_bits,'Parity','None','StopB
its',1,'Terminator','CR/LF');
fclose(s);
fopen(s);

date = floor(now);
%% UDP Initialization
u=udp('192.168.1.115',40185,'LocalPort',51124,'InputBufferSize',144,'ReadAsyn
cMode','continuous');

fopen(u);

%% General config data getter
%     error_count = 0;
%     config_data(2)=1;
%     while(size(config_data,1)<150)
%         while(config_data(2)==1) %gets configuration frame
%             flushinput(u);
%             [config_data,~,msg]=fread(u);
%             if(isequal(msg,'')) %error handling
%             else
%                 clear config_data;
%                 config_data(2)=1;
%                 error_count=error_count+1%#ok
%             end
%         end
%     end
%     config_data(1:14)=[]; % gets rid of non-configuration config_data
%     config_data(1)=[];
%     Frac_time=config_data(1)*2^16+config_data(2)*2^8+config_data(3);
%     config_data_format=config_data(24)*2^8+config_data(25);
%     Phasor_number=config_data(26)*2^8+config_data(27);
%     Phasor_conversion(1:Phasor_number,1) = 0;
%     j = 1;
%     for i=1:Phasor_number
%         Phasor_conversion(i,1)=(config_data(481+4*(i-
1))*2^16+config_data(482+4*(i-1))*2^8+...
%             config_data(483+4*(i-1)))*10^-5; %+config_data(480+4*(i-
1))*2^32
%     end

%% Hardcoded config data

Frac_time = 16777215;
config_data_format = 4;
Phasor_number = 12;
Phasor_conversion = [0.00458; 0.00458; 0.00458; 0.00458; ...
                 0.00107; 0.00107; 0.00107; 0.00107; 0.00107; 0.00107;
0.00107; 0.00107];


%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Initializing variables %%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hours = .05;
count = 1;
k = 1;
j = 1;
delay = 0;
flag = 0;
UDP_dat = zeros(72,1);
day = floor(now);
minutes = hours*60;
index = round(minutes*60*10);
serial_angle   = zeros(1,index);
serial_time    = zeros(1,index);
soc_time_stamp = zeros(1,index);
Frac_sec       = zeros(1,index);
UDP_time       = zeros(1,index);
UDP_angle      = zeros(1,index);
unix_epoch = datenum(1970,1,1,0,0,0);

%% Setting up plots to be plotted.
figure(1);
hold on;
subplot(2,1,1);
lHandle_serial = line(nan,nan);
lHandle_udp = line(nan,nan);
grid on;
subplot(2,1,2);
lHandle_diff = line(nan,nan);
title('serial-UDP');
grid on;

%% Setting up plotting variables
```

```
data_samples = 900;
x_serial = 1:data_samples;
x_udp = 1:data_samples;
x_diff = 1:data_samples;
y_serial = zeros(1,data_samples);
y_udp = zeros(1,data_samples);
y_diff = zeros(1,data_samples);

%% Actual reading of data
while (count<index+1);
        serial_dat = fgetl(s);
        [tmp1,~,msg] = fread(u);
        UDP_dat = tmp1;
        flushinput(u);
        serial_angle(count) = str2double(serial_dat(1:7));
        serial_time(count) =
str2double(serial_dat(8:9))*3600+str2double(serial_dat(10:11))*60+str2double(
serial_dat(12:17));
        if(size(UDP_dat,1)>20)
            while(serial_angle(count)>180)
                serial_angle(count) = serial_angle(count)-360;
            end
            soc_time_stamp(count) =
UDP_dat(7)*2^24+UDP_dat(8)*2^16+UDP_dat(9)*2^8+UDP_dat(10);
            Frac_sec(count)=UDP_dat(12)*2^16+UDP_dat(13)*2^8+UDP_dat(14);
            %%% this is the time stamp
            UDP_time(count) =
round((mod(soc_time_stamp(count),86400)+Frac_sec(count)/Frac_time)*10)/10;
            if(count>1)
                if(UDP_time(count)==UDP_time(count-1))
                    while(1)
                        [UDP_dat,~,msg] = fread(u);
                        if(size(UDP_dat,1)>20)
                            soc_time_stamp(count) =
UDP_dat(7)*2^24+UDP_dat(8)*2^16+UDP_dat(9)*2^8+UDP_dat(10);

Frac_sec(count)=UDP_dat(12)*2^16+UDP_dat(13)*2^8+UDP_dat(14);
                            %%% this is the time stamp
                            UDP_time(count) =
round((mod(soc_time_stamp(count),86400)+Frac_sec(count)/Frac_time)*10)/10;
                            break;
                        end
                    end
                end
            end
            for i=1:Phasor_number
                x_real = typecast(uint16(UDP_dat(18+4*(i-1))+UDP_dat(17+4*(i-
1))*2^8),'int16');
                x_imag = typecast(uint16(UDP_dat(20+4*(i-1))+UDP_dat(19+4*(i-
1))*2^8),'int16');
                %UDP_mag(count,i) =
sqrt(double(x_real)^2+double(x_imag)^2).*Phasor_conversion(i,1);
                UDP_angle(i,count) =
atan2(double(x_imag),double(x_real))/pi*180;
            end
            if(UDP_angle(2,count) == 0)
                flushinput(u);
            end
            disp(count/600/minutes*100);
            disp(serial_angle(count));
            %% Does the real time plotting
            delay = delay+1;
            flag = 1;
            if(k<=data_samples)
                y_serial(k) = serial_angle(count);
                y_udp(k) = UDP_angle(2,count);
                y_diff(k) = y_serial(k)-y_udp(k);
                if(y_diff(k)>180)
                   y_diff(k) = y_diff(k)-360;
                elseif(y_diff(k)<-180)
                   y_diff(k) = y_diff(k)+360;
                end

set(lHandle_serial,'Xdata',x_serial,'Ydata',y_serial,'Color','red');
                set(lHandle_udp,'Xdata',x_udp,'Ydata',y_udp);
                set(lHandle_diff,'Xdata',x_diff,'Ydata',y_diff);
                drawnow;
            else
                k = 0;
            end
            if(UDP_angle(2,count)==0)
                if(count>1)
                    count = count - 1;
                    if(k>0)
                        k = k - 1;
                    end
                end
            end
            k = k + 1;
            count = count + 1;
            flushinput(u);
        end
end

temp1 = serial_time;
temp2 = serial_angle;
temp3 = UDP_time;
temp4 = UDP_angle;

%% Post-Processing on UDP/Serial Data to align GPS data and delete extra data
from UDP.
serial_time(imag(serial_time)~=0) = [];
serial_time(1) = [];
serial_angle(1) = [];
K = find(serial_time == 0);
serial_time(K) = [];
serial_angle(K) = [];
K = find(UDP_time == 0);
UDP_time(K) = [];
UDP_angle(:,K) = [];

i = 2;
while (i < size(UDP_time,2))
    if(UDP_time(i) == UDP_time(i-1))
```

```
        UDP_time(i) = [];
        UDP_angle(:,i) = [];
        i = i - 1;
    end
    i = i + 1;
end
i = 1;
ind = size(UDP_time,2);
if ind == 0
    disp('BARKASDFLKJASDLKFHASDKFK  BARK BARK');
end
counting = 0;

% Aligns the GPS data of the serial data and the UDP data.
while (i < ind)
    while(UDP_time(i) ~= serial_time(i))
        if(UDP_time(i) > serial_time(i))
            serial_time(i) = [];
            serial_angle(i) = [];
        else
            UDP_time(i) = [];
            UDP_angle(:,i) = [];
        end
        if(i > size(UDP_time,2) || i > size(serial_time,2))
            break;
        end
        counting = counting + 1;
        if(counting > 50)
            disp('Warning');
            break;
        end
    end

    if(size(serial_time,2) > size(UDP_time,2))
        ind = size(UDP_time,2);
    else
        ind = size(serial_time,2);
    end
    i = i + 1;
end

if(serial_time(size(serial_time,2))>UDP_time(size(UDP_time,2)))
    while(size(UDP_time,2) ~= size(serial_time,2))
        serial_time(size(serial_time,2)) = [];
        serial_angle(size(serial_angle,2)) = [];
    end
elseif(serial_time(size(serial_time,2))<UDP_time(size(UDP_time,2)))
    while(size(UDP_time,2) ~= size(serial_time,2))
        UDP_time(size(UDP_time,2)) = [];
        UDP_angle(:,size(UDP_angle,2)) = [];
    end
end

% Computes the difference between the serial & UDP data.
p = serial_angle;
q = UDP_angle(2,:);
```

```
errorA = p-q;
errorA(errorA<-180) = errorA(errorA<-180)+360;
errorA(errorA>180) = errorA(errorA>180)-360;


error1 = unwrap(p)-unwrap(UDP_angle(1,:));
% errorA = unwrap(p)-unwrap(UDP_angle(2,:));
errorB = unwrap(p)-unwrap(UDP_angle(3,:));
errorC = unwrap(p)-unwrap(UDP_angle(4,:));

% Computes the median of the difference.
o = ones(1,size(serial_time,2));
o = o*median(errorA);


%% Computes the system omega from the UDP and Encoder data using a 5-point
derivative
for i = 3:size(UDP_angle(2,:),2)
    if(i<size(UDP_angle(2,:),2)-2)
        freq5UDP(i) = (-UDP_angle(2,i+2)+8*UDP_angle(2,i+1)-8*UDP_angle(2,i-
1)+UDP_angle(2,i-2))/1.2;
        freq5Encoder(i) = (-serial_angle(i+2)+8*serial_angle(i+1)-
8*serial_angle(i-1)+serial_angle(i-2))/1.2;
    end
    %freq2(i) = (UDP_angle(2,i)-UDP_angle(2,i-1))/.1;
end

% Tries to delete some of the noisier frequency data.
freq5UDP(abs(freq5UDP)>10) = [];

%% Plotting the final graphs.
figure;
subplot(2,1,1);
title('Delta Comparisons');
plot(serial_time/86400,p,'red',UDP_time/86400,q);
datetick('x','HH:MM:SS.FFF','keepticks');
ylabel('Machine delta');
xlabel('Time');
legend('adjusted delta','UDP Phase A')
subplot(2,1,2);
plot(serial_time/86400,errorA,serial_time/86400,o);%,serial_time/86400,error1
);%,serial_time/86400,errorB,serial_time/86400,errorC);
ylabel('Difference delta');
xlabel('Time');
legend('DiffA','mean');%,'DiffB','DiffC');
datetick('x','HH:MM:SS.FFF','keepticks');
%
plot(UDP_time/86400,UDP_angle(1,:),UDP_time/86400,UDP_angle(2,:),UDP_time/864
00,UDP_angle(3,:),UDP_time/86400,UDP_angle(4,:));
% ylabel('PMU Theta');
% xlabel('Time');
% legend('V1','VA','VB','VC')
% datetick('x','HH:MM:SS.FFF');

figure
subplot(2,1,1);
plot(freq5UDP);
```

```
hold on;
freq5Encoder(abs(freq5Encoder)>10) = [];
plot(freq5Encoder,'green');
legend('PMU','Encoder');
title('Frequency difference from 60');

%% Clears the UDP & Serial Ports

if(exist('u','var'))
    flushinput(u);
    fclose(u);
    delete(u);
    clear u;
end
if(exist('s','var'))
    flushinput(s);
    fclose(s);
    delete(s);
    clear s;
end
```

# Appendix C – Rotor Angle Microprocessor Code

## computerUART.h

```
/*
 * File:   computerUART.h
 * Author: Jacques
 *
 * Created on October 14, 2013, 5:52 PM
 */
#include <plib.h>

void init_UART_JE(UART_MODULE, int);
void init_GPS_JF(UART_MODULE, int);
void test_GPS_UART(void);
inline int ReadGPS_UARTData(char*);
void SendUARTData(const char *buffer, UINT32 size, UART_MODULE id);
```

## HardwareProfile.h

```
// Clock constants

#define SYS_CLOCK_FREQ       80000000
#define GetSystemClock()     (SYS_CLOCK_FREQ)
#define GetPeripheralClock()  (SYS_CLOCK_FREQ / 8)  // FPBDIV = DIV_8
#define GetInstructionClock() (SYS_CLOCK_FREQ)
#define I2C_CLOCK_FREQ       100000


// I2C constants
#define LCD_I2C_BUS        I2C1          // Channel 1
#define LCD_ADDRESS        0x48           // LCD address
```

## LCD.h

```
#ifndef LCD_H
#define LCD_H

void initLCD();
BOOL lcdString(const char string[]);
BOOL lcdInstruction(const char string[]);

#endif
```

## CRC.h

```
int ComputeCRC(unsigned char *Message, unsigned char MessLen);
```

## SPI.h

```
#include <plib.h>

#ifndef SPI_H
#define SPI_H

void initSPI();

void SPIio(SpiChannel chn, unsigned int sendLen, UINT32* sendBuf,
        unsigned int recvLen, UINT32* recvBuf);

#endif
```

## computerUART.c

```
inline int ReadGPS_UARTData(char *time)
{
   static unsigned char data[17];
   int count;
   int error_count = 0;
   int error = 0;
   int itCount = 0;
   UART1ClearAllErrors();
   while(1) {
     error = UART1GetErrors();
     if(error){
        error = 0;
        error_count++;
        UART1ClearAllErrors();
     }
     else{
         count = 0;
         while(count<17){
            if((uartReg[UART1]->sta.reg & _U1STA_URXDA_MASK)){
               data[count]=UARTGetDataByte(UART1);
               *(time+count) = data[count];
               count++;
            }
         }
         if(data[0]=='$'){
            if(data[1]=='G'){
               return itCount;
            }
         }
         else{
            itCount++;
         }
     }
```

```
    }
    return;
}

void init_UART_JF(UART_MODULE uartPort, int baudRate) //UART2 on JF (GPS)
{
    PORTSetPinsDigitalIn(IOPORT_F,BIT_12|BIT_4|BIT_13); //top row of JF
    PORTSetPinsDigitalOut(IOPORT_F,BIT_5);
    PORTClearBits(IOPORT_F,BIT_4|BIT_5|BIT_12|BIT_13);

    UARTConfigure(uartPort,UART_ENABLE_PINS_TX_RX_ONLY);
    UARTSetFifoMode(uartPort,UART_INTERRUPT_ON_RX_NOT_EMPTY);
    UARTSetLineControl(uartPort, UART_DATA_SIZE_8_BITS | UART_PARITY_NONE | UART_STOP_BITS_1);
    UARTSetDataRate(uartPort, GetPeripheralClock(), baudRate);
    UARTEnable(uartPort, UART_ENABLE_FLAGS(UART_PERIPHERAL | UART_RX | UART_TX));
    INTEnable(INT_U2TX, INT_DISABLED);
    INTEnable(INT_U2RX, INT_DISABLED);
    return;
}

void init_UART_JE(UART_MODULE uartPort, int baudRate) //UART1 on JE (computer)
{
    PORTSetPinsDigitalOut(IOPORT_F,BIT_8); //Pin 2 on JE
    PORTSetPinsDigitalIn(IOPORT_D,BIT_15);
    PORTSetPinsDigitalIn(IOPORT_F,BIT_2);
    PORTClearBits(IOPORT_F,BIT_8 | BIT_2);
    PORTClearBits(IOPORT_D, BIT_15);

    UARTConfigure(uartPort,UART_ENABLE_PINS_TX_RX_ONLY);         // Use only RX and TX pin for the UART (no
handshake)
    UARTSetFifoMode(uartPort, UART_INTERRUPT_ON_RX_NOT_EMPTY); // Set the interrupt trigger condition for
RX and TX
    UARTSetLineControl(uartPort, UART_DATA_SIZE_8_BITS | UART_PARITY_NONE | UART_STOP_BITS_1); // Set
the UART exchange protocol as 8-N-1
    UARTSetDataRate(uartPort, GetPeripheralClock(), baudRate);       // Set the desired baud rate (see the #define
on the top)
    UARTEnable(uartPort, UART_ENABLE_FLAGS(UART_PERIPHERAL | UART_RX | UART_TX));    // Enable the UART
module
    INTEnable(INT_U1TX, INT_DISABLED);
    INTEnable(INT_U1RX, INT_DISABLED);
    return;
}



void SendUARTData(const char *buffer, UINT32 size, UART_MODULE id)
{
    int debug_count=0;
    while(size){
        while(!UARTTransmitterIsReady(id));
        debug_count++;
        UARTSendDataByte(id, *buffer);
        buffer++;
        size--;
    }
    while(!UARTTransmissionHasCompleted(id));
    return;
}
```

## CRC.c

```
/* Compute CRC-CCITT. *Message is a pointer to the first character in the
message; MessLen is the number of characters in the message (not counting
the CRC on the end) */
unsigned int ComputeCRC(unsigned char *Message, unsigned char MessLen)
{
unsigned int crc=0xFFFF;
unsigned int temp;
unsigned int quick;
unsigned int i;
for(i=0;i<MessLen;i++)
 {
 temp = (crc>>8) ^ Message[i];
 crc <<= 8;
 quick = temp ^ (temp >> 4);
 crc ^= quick;
 quick <<=5;
 crc ^= quick;
 quick <<= 7;
 crc ^= quick;
 }
 return crc;
}
```

## LCD.c

```
#include <plib.h>
#include <string.h>
#include "HardwareProfile.h"
#include "LCD.h"



BOOL StartTransfer( BOOL restart )
{
    I2C_STATUS  status;

    // Send the Start (or Restart) signal
    if(restart)
    {
        I2CRepeatStart(LCD_I2C_BUS);
    }
    else
    {
        // Wait for the bus to be idle, then start the transfer
        while( !I2CBusIsIdle(LCD_I2C_BUS) );

        if(I2CStart(LCD_I2C_BUS) != I2C_SUCCESS)
        {
            DBPRINTF("Error: Bus collision during transfer Start\n");
            return FALSE;
        }
    }

    // Wait for the signal to complete
    do
    {
        status = I2CGetStatus(LCD_I2C_BUS);

    } while ( !(status & I2C_START) );

    return TRUE;
}



BOOL TransmitOneByte( UINT8 data )
{
    // Wait for the transmitter to be ready
    while(!I2CTransmitterIsReady(LCD_I2C_BUS));

    // Transmit the byte
    if(I2CSendByte(LCD_I2C_BUS, data) == I2C_MASTER_BUS_COLLISION)
    {
        DBPRINTF("Error: I2C Master Bus Collision\n");
        return FALSE;
    }

    // Wait for the transmission to finish
    while(!I2CTransmissionHasCompleted(LCD_I2C_BUS));

    return TRUE;
}



void StopTransfer( void )
{
    I2C_STATUS  status;

    // Send the Stop signal
    I2CStop(LCD_I2C_BUS);

    // Wait for the signal to complete
    do
    {
        status = I2CGetStatus(LCD_I2C_BUS);

    } while ( !(status & I2C_STOP) );
}



void initLCD()
{
                const char resetDisplay[]  = "*";   // Equivalent to a cycling the power.
                const char enableDisplay[] = "1e";  // Enables display with the backlight off.

    // Set the I2C baudrate.
    I2CSetFrequency(LCD_I2C_BUS, GetPeripheralClock(), I2C_CLOCK_FREQ);

                // Enable the I2C bus.
    I2CEnable(LCD_I2C_BUS, TRUE);

                // Reset and clear the display.
                lcdInstruction("*");
    lcdInstruction("j");
}



BOOL lcdString(const char string[])
```

```
{
   BOOL success = TRUE;

   // Start the transfer for writing data to the LCD.
   if (StartTransfer(FALSE)) {
            I2C_7_BIT_ADDRESS SlaveAddress;
            I2C_FORMAT_7_BIT_ADDRESS(SlaveAddress, LCD_ADDRESS, I2C_WRITE);
                        if (TransmitOneByte(SlaveAddress.byte)) {
                                    const char* cp;
         // Call TransmitOneByte() for each non-null character in the string.
         // Stop with success set to false if TransmitOneByte() returns failure.
                                    int counter=0;
                  int size = sizeof(string);

                                    while(counter<size)
                                    {
                                                success=TransmitOneByte(string[counter]);
                                                counter++;
                                                if(!success)
                                                {
                                                            StopTransfer();
                                                            return success;
                                                }
                                    }
                                    if (success) {
                                                StopTransfer();
                                    }
                        }
                        else {
                                    success = FALSE;
            }
            }
            else {
                        success = FALSE;
            }
            return success;
}


BOOL lcdInstruction(const char string[])
{
            char instruction[32] = "\x1B[";  // Instruction escape preamble.
            return lcdString(strncat(instruction,string,28));
}
```

## SPI.c

```
#include "SPI.h"

#define SS_PORT IOPORT_F
#define SS_PIN     BIT_12



void initSPI()
{
            //DDPCONbits.JTAGEN = 0;
            PORTSetPinsDigitalOut(SS_PORT,SS_PIN);
            SpiChnOpen(SPI_CHANNEL1,SPI_OPEN_MSTEN|SPI_OPEN_CKE_REV|SPI_CONFIG_MODE32|SPI_CONFIG
_CKP_HIGH,200);
}


void SPIio(SpiChannel chn, unsigned int sendLen, UINT32* sendBuf,
unsigned int recvLen, UINT32* recvBuf)
{
            int sendCount=0;
            while(SpiChnIsBusy(chn)); //makes sure spi isn't working
      SpiChnReadC(chn);//clears out spi receive buffer
            //PORTClearBits(SS_PORT,SS_PIN);
            while(sendCount<sendLen)
            {
                        if(!SpiChnTxBuffFull(chn))
                        {
                                    SpiChnWriteC(chn,*sendBuf);
                                    ++sendCount;
                        }
            while(!SpiChnDataRdy(chn))
            {}
            *recvBuf=SpiChnReadC(chn);//clears out spi receive buffer
            ++sendCount;
      }
}
```

## Main.c

```
#include <plib.h>
#include <stdlib.h>
#include "HardwareProfile.h"


// Configuration Bit settings
//
// SYSCLK = 80 MHz (8MHz Crystal/ FPLLIDIV * FPLLMUL / FPLLODIV)
// PBCLK = 10 MHz
// WDT OFF
#define Timer_core_value 4000000 //400000
#define Timer_1_value 1000000
#define comp_BAUD 50000

#ifndef OVERRIDE_CONFIG_BITS

  #pragma config FPLLMUL  = MUL_20       // PLL Multiplier
  #pragma config FPLLIDIV = DIV_2        // PLL Input Divider
  #pragma config FPLLODIV = DIV_1        // PLL Output Divider (80MHz system clock)
  #pragma config FPBDIV   = DIV_8        // Peripheral Clock divisor (10MHz peripheral clock)
  #pragma config FWDTEN   = OFF          // Watchdog Timer
  #pragma config WDTPS    = PS1          // Watchdog Timer Postscale
  #pragma config FCKSM    = CSDCMD       // Clock Switching & Fail Safe Clock Monitor
  #pragma config OSCIOFNC = OFF          // CLKO Enable
  #pragma config POSCMOD  = EC//MIGHT HAVE TO CHANGE BACK TO HS          // Primary Oscillator
  #pragma config IESO     = OFF          // Internal/External Switch-over
  #pragma config FSOSCEN  = OFF          // Secondary Oscillator Enable
  #pragma config FNOSC    = PRIPLL       // Oscillator Selection
  #pragma config CP       = OFF          // Code Protect
  #pragma config BWP      = OFF          // Boot Flash Write Protect
  #pragma config PWP      = OFF          // Program Flash Write Protect
  #pragma config ICESEL   = ICS_PGx1     // ICE/ICD Comm Channel Select
  #pragma config DEBUG    = OFF          // Debugger Disabled for Starter Kit

#endif // OVERRIDE_CONFIG_BITS

// *****************************************************************************
// *****************************************************************************
// Application Main Entry Point
// *****************************************************************************
// *****************************************************************************
inline unsigned int gray2bin(UINT32);
inline int PTP(int*, unsigned int);
inline void display(int);
inline void toSerial(char*, double);
inline void Initialization();
```

```
int ptpCount = 0;
int main()
{
   ////////////////////////////////////////////////////
   /////////  Variable Declaration  ////////////////////
   ////////////////////////////////////////////////////
   unsigned int debug_timer[60];
   int count = 0;
   int PTPon = 2;
   int secondCount = 0;
   int minuteCount = 0;
   int i = 0;
   int begin_timer = 1;
   unsigned int fix_timer = 4000000;
   int Receive_data= 0;
   unsigned int binary = 0;
   char comp_uart_buffer[19];
   double angle = 0;
   double single_turn_resolution = 4096;
   char UTC_Time[17];
   int second = 0,minute = 0,hour = 0;
   time_t date = time(NULL);
   struct tm *timer = localtime(&t);
   timer->tm_sec -= timer->tm_sec;
   timer->tm_min -= timer->tm_min;
   timer->tm_hour -= timer->tm_hour;

   ///////////////////////////////////////////////////
   //////  Initialization  ///////////////////////////
   ///////////////////////////////////////////////////
   Initialization();
   ///////////////////////////////////////////////////
   //////  Clears communication Channels  ////////////
   ///////////////////////////////////////////////////
   while(UARTReceivedDataIsAvailable(UART1)){
      UARTGetDataByte(UART1);
   }
   while(SpiChnIsBusy(SPI_CHANNEL1)); //makes sure spi isn't working
   SpiChnReadC(SPI_CHANNEL1);//clears out spi receive buffer
   ///////////////////////////////////////////////////
   //////////  Main Program Loop  ////////////////////
   ///////////////////////////////////////////////////
   while(1){
      if(count){
         display(binary);
         while(ReadCoreTimer() < fix_timer);
         SpiChnWriteC(SPI_CHANNEL1,0x00000000);
         WriteCoreTimer(0);
         count++;
         if(count==10){
             count = 0; //should be set to 0 when GPS works
         }
         while(!SpiChnDataRdy(SPI_CHANNEL1));
         Receive_data=SpiChnReadC(SPI_CHANNEL1);
         ////////////////////////////////////////////////////
         //interpret the encoder data into an angle //////////
         ////////////////////////////////////////////////////
         // gets the non-garbage data being sent (7 due to SPI and SSI not being perfectly aligned. 0x80 due to 3
 error bits being ignored and the rest
         // are garbage bits since data from encoder is 27 bits and 32 clock pulses are being sent.
         Receive_data = Receive_data & 0x7FFFFF80;
         Receive_data = Receive_data >> 7;         // shift the bits to make LSB the 0th bit
         ////////////////////////////////////////////////////
         // Gray to binary conversion & angle recovery ////////
         ////////////////////////////////////////////////////
         binary = gray2bin(Receive_data);
         angle = ((double)binary)/single_turn_resolution*360;
         /////////////////////////////////////////////
         //PREPARES SERIAL UART BUFFER    /////////////
         /////////////////////////////////////////////
         toSerial(comp_uart_buffer,angle);
         /////////////////////////////////////////////
         ////  ATTACHES 'GPS' DATA HERE  //////////////
         /////////////////////////////////////////////
         i = 7;
         while(i<17){
            comp_uart_buffer[i]= UTC_Time[i];
            i++;
         }
         if(count)
            comp_uart_buffer[14]='0'+count-1;
         else
            comp_uart_buffer[14]='0'+9;

         /////////////////////////////////////////////
         //  send data over UART here. ////////////////
         /////////////////////////////////////////////
         SendUARTData(comp_uart_buffer,sizeof(comp_uart_buffer),UART2);
         SpiChnReadC(SPI_CHANNEL1);
      }

      else //this section both realigns timing to GPS and also fixes frequency due to offset
            {
         ////////////////////////////////////////////////////
         // POLLS PPS BIT AND LATCHES ENCODER DATA ///////////
         ////////////////////////////////////////////////////
         SpiChnReadC(SPI_CHANNEL1);
         while(PORTReadBits(IOPORT_D,BIT_15)==0);
```

```
SpiChnWriteC(SPI_CHANNEL1,0x00000000); //this takes 52 cycles
debug_timer[secondCount]=ReadCoreTimer();
WriteCoreTimer(0);

while(!SpiChnDataRdy(SPI_CHANNEL1));    // waits for encoder data to be ready
{}
////////////////////////////////////////////////////
//////////////// Gets Ecoder Data ///////////////////
////////////////////////////////////////////////////
Receive_data=SpiChnReadC(SPI_CHANNEL1); // gets encoder data
Receive_data = Receive_data & 0x7FFFFF80;
Receive_data = Receive_data >> 7;         // shift the bits to make LSB the 0th bit

////////////////////////////////////////////////////
// Gray to binary conversion ////////////////////////
////////////////////////////////////////////////////
binary = gray2bin(Receive_data);
angle = ((double)binary)/single_turn_resolution*360;
////////////////////////////////////////////////////
//PREPARES SERIAL UART BUFFER    ////////////////////
////////////////////////////////////////////////////
toSerial(comp_uart_buffer,angle);

if((secondCount%60)==0){
   ReadGPS_UARTData(UTC_Time);
   second = ((UTC_Time[11]-'0')*10) + (UTC_Time[12]-'0');
   minute = ((UTC_Time[9]-'0')*10) + (UTC_Time[10]-'0');
   hour = ((UTC_Time[7]-'0')*10) + (UTC_Time[8]-'0');
}
else{
   second++;
   if(second==60){
      second = 0;
      minute++;
      if(minute==60){
         minute = 0;
         hour++;
         if(hour==24){
            hour = 0;
         }
      }
   }
   UTC_Time[7] = hour/10    + '0';
   UTC_Time[8] = hour%10    + '0';
   UTC_Time[9] = minute/10  + '0';
   UTC_Time[10] = minute%10 + '0';
   UTC_Time[11] = second/10 + '0';
   UTC_Time[12] = second%10 + '0';
```

```
        }
        //////////////////////////////////////////////////////
        //ATTACHES GPS DATA  ////////////////////////////////
        //////////////////////////////////////////////////////
        i = 7;
        while(i<17){
           comp_uart_buffer[i]= UTC_Time[i];
           i++;
        }
        //////////////////////////////////////////////////////
        //send data over UART here. /////////////////////////
        //////////////////////////////////////////////////////
        SendUARTData(comp_uart_buffer,sizeof(comp_uart_buffer),UART2);
        //////////////////////////////////////////////////////
        //TRIES TO IMPLEMENT A CRUDE FORM OF PTP ////////////
        //////////////////////////////////////////////////////
        if(secondCount==60){
           if(begin_timer){
              fix_timer = PTP(debug_timer,fix_timer);
              begin_timer = 0;
           }
           minuteCount++;
           secondCount = 0;
           //turns PTP on every 30 minutes for a 1 minute average.
           if(ptpCount==2){
              PTPon = 29;
              ptpCount = 0;
           }
           if(minuteCount==PTPon){
              minuteCount = 0;
              begin_timer = 1; // turns on the PTP averaging
           }
        }
        secondCount++;
        count++;
           }
    }
    return 0;
}

inline void Initialization()
{
    int GPS_BAUD = 9600;
    ///////////////////////////
    // char speedCommand[] = "$PMTK251,4800*14\r\n"; //this one doesn't change the speed
    // char speedCommand[] = "$PMTK251,19200*22\r\n";
    // char speedCommand[] = "$PMTK251,38400*27\r\n";
     char speedCommand[] = "$PMTK251,57600*2C\r\n";
    // char speedCommand[] = "$PMTK251,115200*1F\r\n";
    char updateRate[] = "$PMTK300,200,0,0,0,0*2F\r\n";
    //char messageType[] = "$PMTK314,5,1,5,1,5,5,5,5,5,5,5,5,5,5,5,5,5,5,5*2C\r\n";
    AD1PCFG = 0xFFFFF;
    initSPI();
    initLCD();
    init_UART_JF(UART2,comp_BAUD);
    init_UART_JE(UART1,GPS_BAUD);

    GPS_BAUD = 57600;
    SendUARTData(updateRate,sizeof(updateRate),UART1);
    SendUARTData(speedCommand,sizeof(speedCommand),UART1);
    UARTSetDataRate(UART1, GetPeripheralClock(),GPS_BAUD);
    return;
}

inline void toSerial(char* charArray, double angle)
{
    int temp2;
    *charArray = '0'+(((int)angle/100));
    *(charArray+1)='0'+(((int)angle%100)/10);
    *(charArray+2)='0'+(((int)angle%10));
    *(charArray+3)='.';
    temp2 = (int)(angle*1000) % 1000;
    *(charArray+4)='0'+(temp2/100);
    *(charArray+5)='0'+(temp2%100/10);
    *(charArray+6)='0'+(temp2%10);
    *(charArray+17)='\r';
    *(charArray+18)='\n';
    return;
}
inline void display(int binary)
{
    int temp1,temp2;
    const char clearDisplay[] = "j";
    char Hexvalue[5];
    temp2 = binary;
    temp1 = temp2 / 1000;
    Hexvalue[0]=temp1+'0';
    temp2 = temp2 % 1000;
    temp1 = temp2 / 100;
    Hexvalue[1]=temp1+'0';
    temp2 = temp2 % 100;
    temp1 = temp2 / 10;
    Hexvalue[2]=temp1+'0';
    temp2 = temp2 % 10;
    Hexvalue[3] = temp2 + '0';
    lcdInstruction(clearDisplay);
    lcdString(Hexvalue);
    return;
}
inline unsigned int gray2bin(UINT32 Receive_data)
{
    unsigned int binary = Receive_data  & 0x800000;
    binary = Receive_data  & 0x800000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x400000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x200000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x100000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x80000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x40000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x20000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x10000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x8000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x4000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x2000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x1000;
    binary |= (Receive_data ^ (binary >> 1)) & 0x800;
    binary |= (Receive_data ^ (binary >> 1)) & 0x400;
    binary |= (Receive_data ^ (binary >> 1)) & 0x200;
    binary |= (Receive_data ^ (binary >> 1)) & 0x100;
    binary |= (Receive_data ^ (binary >> 1)) & 0x80;
    binary |= (Receive_data ^ (binary >> 1)) & 0x40;
    binary |= (Receive_data ^ (binary >> 1)) & 0x20;
    binary |= (Receive_data ^ (binary >> 1)) & 0x10;
    binary |= (Receive_data ^ (binary >> 1)) & 0x8;
    binary |= (Receive_data ^ (binary >> 1)) & 0x4;
    binary |= (Receive_data ^ (binary >> 1)) & 0x2;
    binary |= (Receive_data ^ (binary >> 1)) & 0x1;
    binary = binary & 0xFFF;
    return binary;
}

inline int PTP(int* times, unsigned int fix_timer)
{
    int z;
    int y;
    int ymin;
    unsigned int temp;
    unsigned int timer_storage[60];
    for (z=0;z<60;z++)
    {
        timer_storage[z]=*(times+z);
    }
    for(z=0; z<60; z++){
        ymin = z;
        for (y = z + 1; y<61; y++){
            if(timer_storage[y] < timer_storage[ymin]){
                ymin = y;
            }
        }
        temp = timer_storage[z];
        timer_storage[z]=timer_storage[ymin];
        timer_storage[ymin]=temp;
    }
    if(timer_storage[29]<4000000){
        if(Timer_core_value - timer_storage[29] > 50){
            fix_timer = fix_timer - ((Timer_core_value - timer_storage[29]))/(10*(ptpCount+1));
            ptpCount++;
        }
    }
    else{
        if(timer_storage[29]-Timer_core_value > 50){
            fix_timer = fix_timer + (timer_storage[29]-Timer_core_value)/(10*(ptpCount+1));
            ptpCount++;
        }
    }
    return fix_timer;
}
```