Characterization, Control and Compensation of MEMS Rate and Rate-Integrating Gyroscopes

by

Jeffrey A. Gregory

A dissertation submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Electrical Engineering)
in The University of Michigan
2012

For Jishu

ACKNOWLEDGEMENTS

I would like to thank to my wife Jishu for putting up with me, and my family for raising me. Also, thank you to Dr. Jae Yoong Cho for fabricating the gyroscopes and his insight, and to all of my group mates for their camaraderie. Thank you to Robert Gordenker for making our lab space the best in EECS. Thank you to Trasa Burkhardt for being the best secretary a student could ask for. Thank you to Dr. Becky Peterson for taking care of all those parts of research that no one wants to deal with. Thank you to my committee for their helpful comments. And thank you to my adviser Professor Khalil Najafi for supporting my work and pushing me to make it the best I can.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

LIST OF ABBREVIATIONS

A_g	angular gain
ADPLL	all-digital phase-locked loop
ARW	angle random walk
ASIC	application-specific integrated-circuit
CING	Cylindrical Rate-Integrating Gyroscope
COTS	commercial off the shelf
CVG	Coriolis vibratory gyroscope
DSP	digital signal processor
INS	inertial navigation system
IMU	inertial measurement unit
LDV	Laser Doppler Vibrometry
LSB	least significant bit
MAV	micro aerial vehicle
MEMS	micro electrical mechanics systems
PID	proportional, integral, differential
RIG	rate integrating gyroscope
RLG	rate-laser gyroscope
RRW	rate random walk
SDR	software-defined radio
SNR	signal-to-noise ratio

SOG	silicon on glass
URAT	Universal Resonator Analysis Tool
ZRO	Zero-rate offset

ABSTRACT

Characterization, Control and Compensation of MEMS Rate and Rate-Integrating Gyroscopes

by

Jeffrey A. Gregory

Chair: Khalil Najafi

Inertial sensing has important applications in navigation, safety, and entertainment. Areas of active research include improved device structures, control schemes, tuning methods, and detection paradigms. A powerful and flexible characterization and control system built on commercial programmable hardware is especially needed for studying mode-matched gyroscopes and rate-integrated gyroscopes. A gyroscope can be operated in a mode-matched rate-mode for increased sensitivity or rate-integrating mode for greatly increased dynamic range and bandwidth, however control is challenging and the performance is sensitive to the matching of the modes.

This thesis proposes a system built on open and inexpensive software-defined radio (SDR) hardware and open source software for gyroscope characterization and control. The characterization system measures ring-down of devices with damping times and automatically tunes the vibration modes from over 40 Hz mismatch to better than 100 mHz in 3 minutes. When used for rate-gyroscope operation the system provides an FPGA implementation of rate gyroscope control with amplitude, rate and quadrature closed-loop control in the SDR hardware which demonstrates

400% improvement in noise and stability over open-loop operation. The system also operates in a RIG mode with hybrid software/firmware control and demonstrates continuous operation for several hours, unlike previous systems which are limited by the gyroscope ring-down time. The hybrid mode also has a simulation module for development of advanced gyroscope control algorithms. Advanced controls proposed for RIG operation show over 1000% improvement in effective frequency and damping mismatch in simulation and 25% reduction in drift due to damping mismatch in a test RIG. By tuning the compensation, the drift can be reduced by almost 90%, with worst case drift decreased to -41°/s and RMS drift to -21°/s. Harmonic analysis of the anisotropy in a rate-integrating gyroscope measured with this control system is presented to guide development of new error models which will further improve performance.

CHAPTER I

Introduction

Gyroscopes are instruments to measure rotation and exist in many forms. The small gyroscopes found in commercial devices such as cell-phones and tablets are micro electrical mechanics systems (MEMS) vibratory gyroscopes. Low-price and limited performance requirements make MEMS gyroscopes ideal for these applications. The market for standalone MEMS gyroscopes however is expected to be begin to shrink over the next few years, Figure 1.1, as the focus shifts to systems that combine multiple gyroscopes and accelerometers. These combined systems can form the basis of an inertial navigation system (INS) at the low end for gaming, at the mid-range to improve the responsiveness of cell-phone GPS, or at the very high-end for tracking the movements of rescue workers in a building, or guiding micro-aerial vehicles and smart projectiles. However, for MEMS gyroscopes to gain more traction in the inertial navigation market, performance needs to improve. The design goals of a MEMS INS are that it should be small, meaning small sensors and highly integrated electronics; have very low angle drift, which requires devices with low stress and high quality factor; wide-dynamic range, being able to measure well at both low and high rotation rates; and wide bandwidth to match the maneuverability of small vehicles. The work in this thesis addresses the challenges of developing next generation inertial sensors and inertial navigation systems at the system level with software for

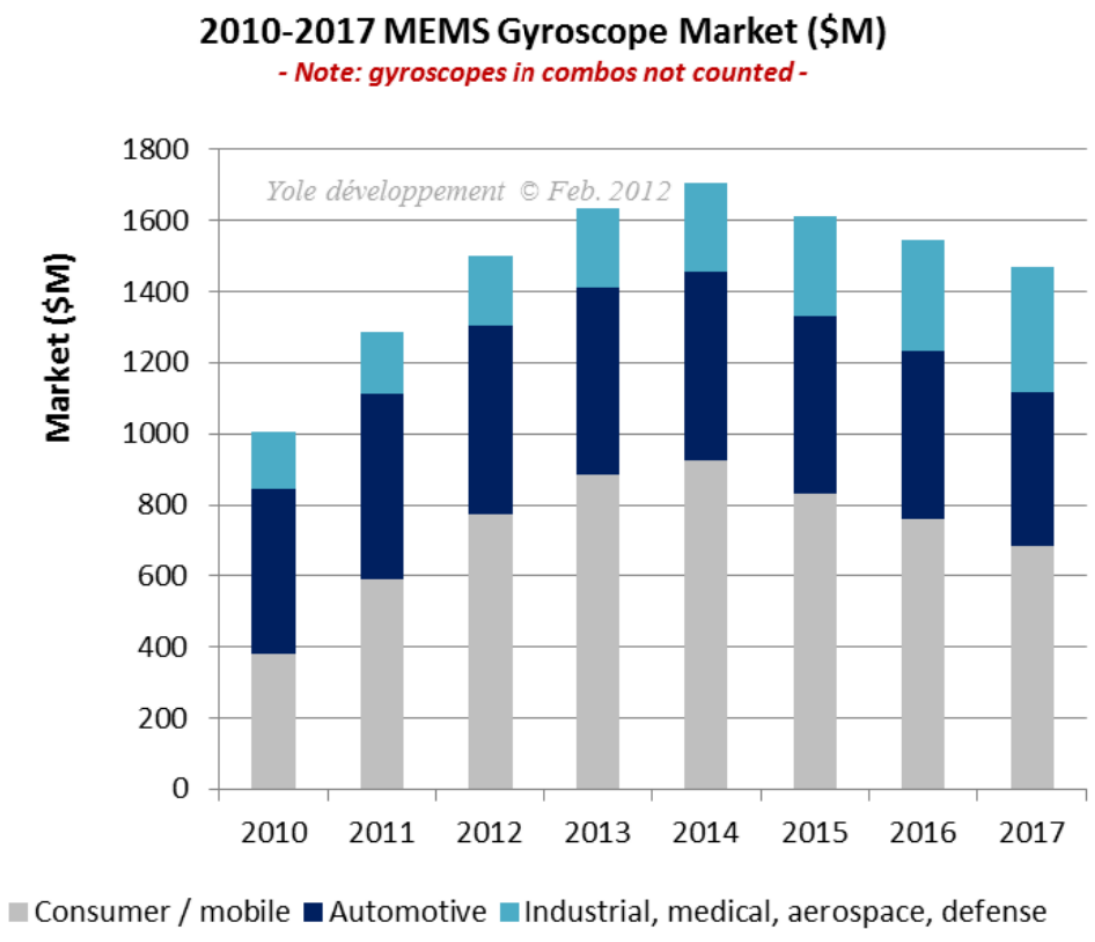


Figure 1.1: A projection of the MEMS gyroscope market size. From [1], this predicts that the market for standalone MEMS gyroscopes will begin to shrink as the market moves towards sensors with multiple degrees of freedom.

system level modeling, the development of a powerful and flexible characterization and control system, and new algorithms for control of MEMS gyroscopes in angle or rate-integrating mode.

Performance metrics for gyroscopes include rate noise (angle random walk (ARW)), angle noise (rate random walk (RRW)), bias stability or drift, bandwidth, and dynamic range which are explained in more detail in Section 2.4. While bandwidth is often thought of as the response to a vibratory rotation it is also critical for the response to a step function or impulse in the rate. These kinds of rates are encountered when small vehicles such as a micro aerial vehicle (MAV) or people make sharp

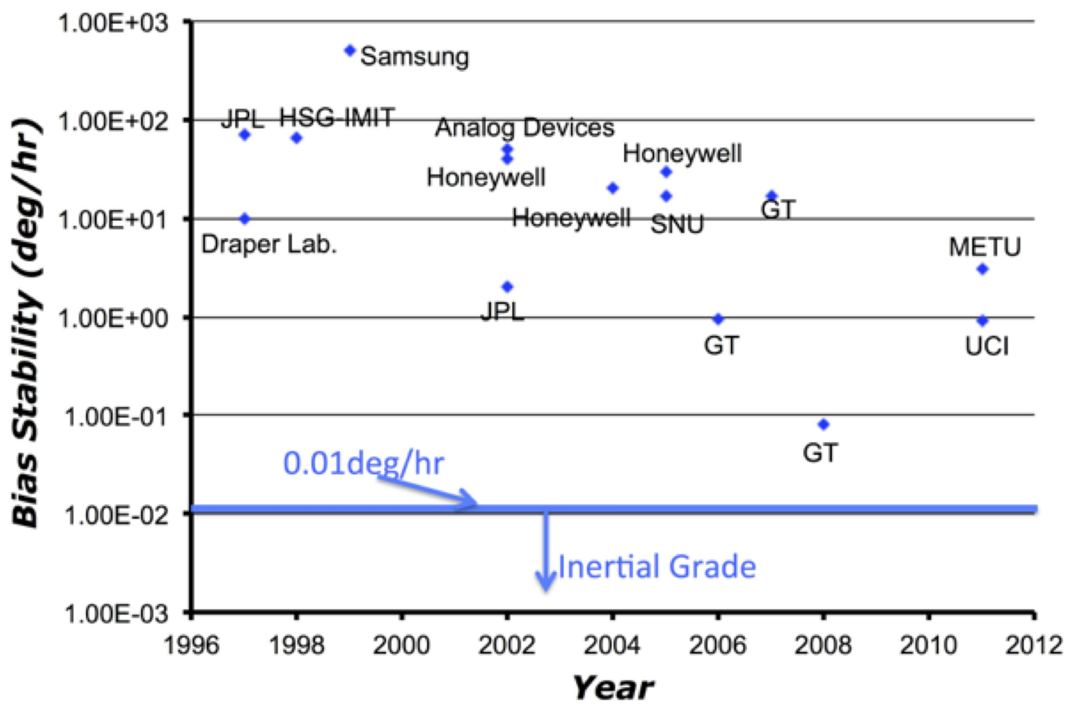


Figure 1.2: Progression of MEMS gyroscope bias stability. From [3].

turns. Dynamic range is the difference between the lowest and highest measurable rate. Very high rates of rotation are also more common in a small vehicle. For gyroscopes with low noise and drift, a very high dynamic range would be $1800°/s$, which is only about 6 rotations per second. Typically for very high rate applications, there is a large performance sacrifice in terms of rate noise and bias stability [2]. Advancing gyroscope technology is typically expressed in terms of bias stability, which has units of $°/Hr$, as in Figure 1.2. Low term noise like bias stability and RRW are important for a large aircraft where bandwidth is usually only a few Hertz and dynamic range may only need to be tens of degrees per second. A smaller unmanned vehicle can require a bandwidth of kilohertz and a dynamic range of several thousands of degrees per second.

Gyroscopes can be characterized as rate or rate-integrating gyroscopes depending on whether they measure the rotation rate or the rotation angle (which is the integrated rate). Any device capable of being operated as a rate-integrating gyroscope can also be operated as a rate gyroscope, and in fact the only commercial vibratory rate-integrating gyroscope, the Northrup Grumman HRG [4] is operated as a rate gyro at low rates and switches to a rate-integrating mode when the rate increases.

There are several advantages to rate-integrating operation of a gyroscope. RIG theoretically have nearly infinite bandwidth and dynamic range, which makes them attractive in applications such as ultra-miniaturized autonomous vehicles, pico-satellites, and tactical guidance systems. The downsides of RIG are increased angle drift at low rates relative to the same gyroscope operated in rate mode and increased control complexity. The error in a rate-integrating gyroscope is described in more detail in Section 5.4, briefly though the angle drift in a RIG is proportional to the variation of the stiffness and quality factor (or damping) across the device (anisotropy), and MEMS devices tend to have worse relative anisotropy than meso-scale devices. The gyroscope will not integrate rotation less than the angle drift rates, so the worse the drift the higher the applied rotation should be before operating the gyroscope in rate-integrating mode. A control system to compensate for the drift due to anisotropy will allow rate-integrating MEMS gyroscopes to switch from rate to rate-integrating modes at lower rates, improving overall linearity and performance.

Research groups developing gyroscopes often focus on fabricating devices or theoretical work on control algorithms and lack the expertise to implement effective readout and control hardware. Much of the control work that is done is difficult to replicate because the hardware used is not available commercially and it takes too long to develop new hardware locally. In this thesis, a characterization and control system for MEMS gyroscopes built on commercially available hardware is proposed. The hardware used was developed for software-defined radio applications, but is a powerful and flexible platform suitable for many applications. The system provides characterization tools optimized for MEMS gyroscopes, a resource efficient rate gyro-

Publication	Control	Tuning	Error Compensation	Rate-Integrating	Tuning-Time
UC Irvine [5]	SIMULINK	None	Quadrature suppression.	No	-
UC Irvine [6]	None	electric mode matching	High Q	Yes	Unknown
JPL [7]	FPGA	-	-	Rate	-
UC Irvine [8]	DSP	-	-	Rate	-
Northrop-Grumman [4]	Proprietary	Mechanical polishing, electric mode matching	High Q, quadrature suppression.	Yes	Unknown
UC Berkeley [9]	Simulation	-	Frequency matching, static damping mismatch	Yes	-
This Work	FPGA + Software	Electric mode matching	Quadrature suppression, dynamic damping and frequency mismatch	Yes	1s

Table 1.1: Comparison of programmable and rate-integrating gyroscope control systems.

scope control implemented in an FPGA and suitable for multiple device integration, and a hybrid software-firmware mode for rate and rate-integrating operation of a MEMS gyroscope.

The rate control presented in this work is implemented in a very small FPGA and is optimized to reduce the resource utilization. The controls are also pipelined with the pipeline running much faster than the response time of typical gyroscopes. This leaves a large amount of available time to process additional devices. The same control, and potentially the analog front-end, could be shared between several devices with no significant increase in the hardware cost or complexity per additional device. This approach allows more compact system integration and could even offer performance improvements in some situations where signals to closely packed devices would otherwise interfere with each other.

1.1 Review of Gyroscope Control Platforms

Control of rate gyroscopes is covered in detail in Chapter IV RIG in Chapter V. This section reviews the systems that have been published for working with MEMS gyroscopes which are relatively generic. Work on programmable gyroscope control systems and rate-integrating gyroscope operation is summarized in Table I. The table includes the control method which indicates how controls are implemented, how tuning is performed, any additional error compensation techniques, whether the system is used for rate-integrating operation, and how long tuning takes. The programmable hardware for some of the work in the table, especially [7, 8], is similar to the hardware used in this work. However, that hardware is not commercially available and does not have the software to support all of the characterization, tuning, and control functions presented in this work.

Control of a rate-gyroscope does not require a programmable system. Rate gyroscope control is well suited to analog implementations and is popular in commercial products [10]. Digital rate gyroscope control is emerging however as digital circuitry becomes cheaper. Potential advantages of digital control include high-order filters, avoiding the 1/f regime by directly digitizing signals at the resonance frequency [11], combining the control of multiple devices for an inertial measurement unit (IMU), and the ability to implement more complicated characterization and compensation techniques than is practical in analog circuits. Several programmable gyroscope control platforms have been previously presented in the literature. The most significant limitation of these systems is that they do not use commercially available hardware, so significant effort is required to recreate them.

An FPGA-based rate gyroscope control is presented in [7] and [12]. The control is closed-loop on both the rate and drive axis, but does not include a quadrature control. Although This hardware could implement the controls presented in Chapter IV, it is not commercially available. There are many filters in this control topology,

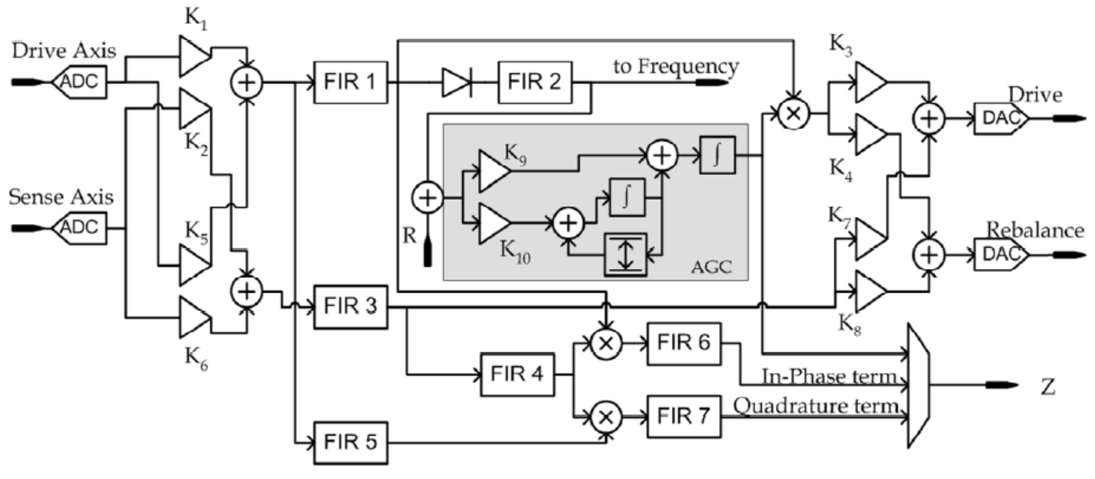


Figure 1.3: Diagram of FPGA based control and characterization system. This closed-loop control was implemented in [7] with the closed-loop frequency extraction in [12], from [12]

each of which is a separate FIR filter. Although each filter is not very high order, there many of them and they will use much more FPGA resources than the method in Chapter IV, making it less appealing for applications with multiple sensors.

A DSP based control is easier to program and potentially lower power and higher performing than an FPGA system. A programmable system based on a DSP is presented in [8] and illustrated in Figure 1.4. If this system were commercially available, it would make an attractive alternative to the hardware platform for the software system presented in this thesis. Currently, it lacks the capability for characterization, tuning, and rate-integrating control presented here. There does not appear to be any software framework available for this hardware either to ease development of the characterization and control system. The features of the system in [8], defined by the components used, are summarized in Table 1.1. To work with higher frequency gyroscopes, one could use higher-speed ADCs. Having to implementing some of the very delay sensitive features, such as the PLL and feed-through compensation, in a DSP instead of an FPGA may still limit performance and will definitely limit the top frequency. There are commercially available development boards with both a DSP

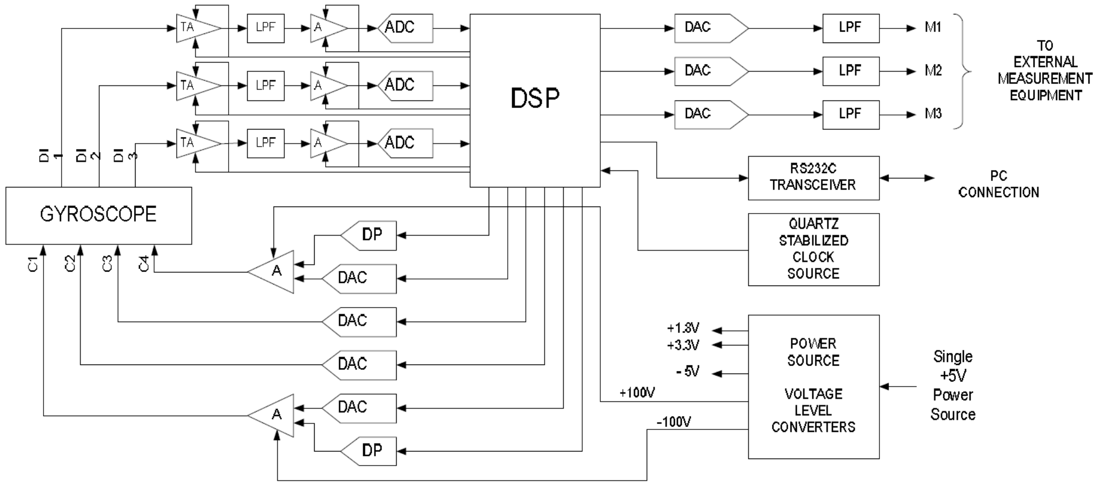


Figure 1.4: Diagram of a DSP based control system. This control system implemented in [8] uses a DSP to implement the control functions instead of an FPGA or analog control.

DSP controller TI TMS320F2812	32-bit 150 MIPS
ADC and DAC conversion update rate	100 kHz
2 DAC AC carrier channels	10 Vp-p, 0.15 mV step
2 DAC AC+DC actuation channels	100 V, 0.8 mV step
3 DAC monitor channels	10 Vp-p, 0.15 mV step
3 differential ADC with transimpedance amplifiers	110 MΩ gain, 18-bit conversion

Table 1.2: Operational parameters of a DSP based control system. Presented in [8]

and FPGA, but without high-voltage handling capabilities. The performance of the proposed system is not provided and the on-board voltage generators are a potential source of interference.

1.1.1 Rate-integrating gyroscope control systems

Recently there has been renewed interest in MEMS RIGs. Control of meso-scale RIG has been studied since the 1960's. The earliest RIG controls were mixed signal [13]. Drive signals were applied at $\frac{1}{2}$ of the resonance frequency and multiplexed in time. These controls do not directly address the drift due to damping anisotropy. The macro-scale gyroscope is mechanically polished to reduce anisotropy, but this method is not practical for MEMS gyroscopes and still does not produce drift on par with rate mode operation.

One of the largest challenges of controlling a nearly ideal RIG is to sustain the oscillation without biasing the angle measurement. In [6], the gyroscope is the most ideal MEMS gyroscope to date and is operated as a RIG by allowing the gyroscope to freely oscillate, without a sustaining loop. The angle is read out in real time using the control system in Figure 1.5. There is no closed-loop control of the oscillation amplitude or compensation for frequency or damping mismatch. The gyroscope used with this readout has a very high damping time of 173 s, which limits the test time to roughly 10 minutes. The advantage of this method is that the gyroscope can be rotated at very high rates without any non-linearity introduced by delay in the control system. The damping mismatch of 0.5% of 173 s is much better than any other reported in literature for a MEMS gyroscope, but the drift is still around 10 $^{\circ}/Hr$ which is much higher than would be expected from the same device operated as a rate gyroscope.

In [14], PLLs are proposed to generate the signals to sustain the oscillation of a rate-integrating gyroscope as in Figure 1.6. Although the system seems simple, the operation can be difficult to follow. If the standing wave of the gyroscope is described as two traveling waves however, the two loops lock to each of the standing waves, and the combined output signals match the standing wave on the gyroscope. This approach is appealing since it is an all-analog method of sustained RIG operation, however there is no compensation for gyroscope errors, and this method is particularly sensitive to gain errors in the drive and readout. Also, any significant level of stiffness anisotropy will keep the gyroscope from working in rate-integrating mode. This control has not been experimentally demonstrated, since MEMS gyroscopes had far too much mismatch to operate with this control.

A programmable control system for RIG control based on Labview SIMULINK was proposed in [5] and included a method to compensate for quadrature error Figure

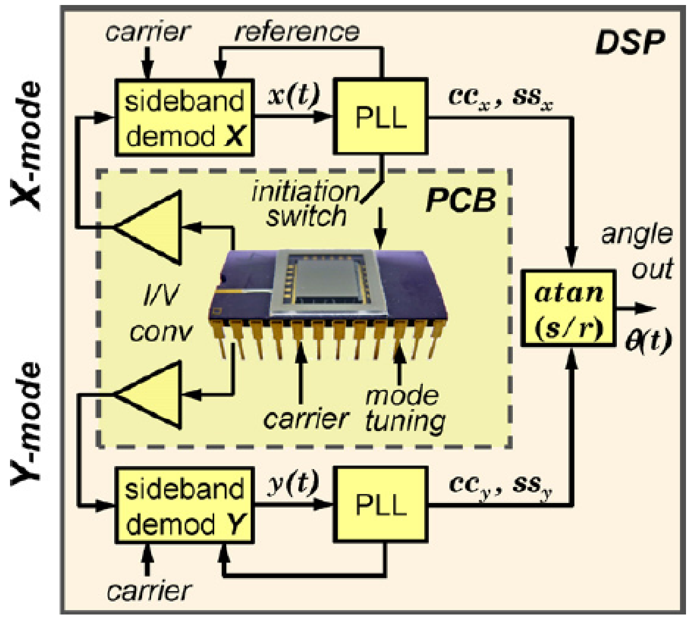


Figure 1.5: Diagram of Freely Oscillating RIG Readout. This readout, from [6], allows real-time sensing of the gyroscope angle, but does not sustain the oscillation and so the measurement duration is limited by the gyroscope ring-down time.

1.7. The quadrature error was extracted using the relationship [15]

$$H = x\dot{y} - y\dot{x} \tag{1.1}$$

where x and y are the displacements and $\dot{x}, \dot{y}$ are the velocities extracted from the sensed signals. The method of extracting the displacement and velocity is not explicitly given. The control law for quadrature, based on [16], is

$$\begin{bmatrix} F_x \\ F_y \end{bmatrix} = -\gamma_1 H S^T \begin{bmatrix} x \\ y \end{bmatrix} \tag{1.2}$$

where γ_1 is the control gain and S^T is a skew symmetric matrix, presumably to compensate for electrode misalignment. The measured angular momentum (quadrature error) was reduced by 31%. This degree of compensation was not sufficient to allow

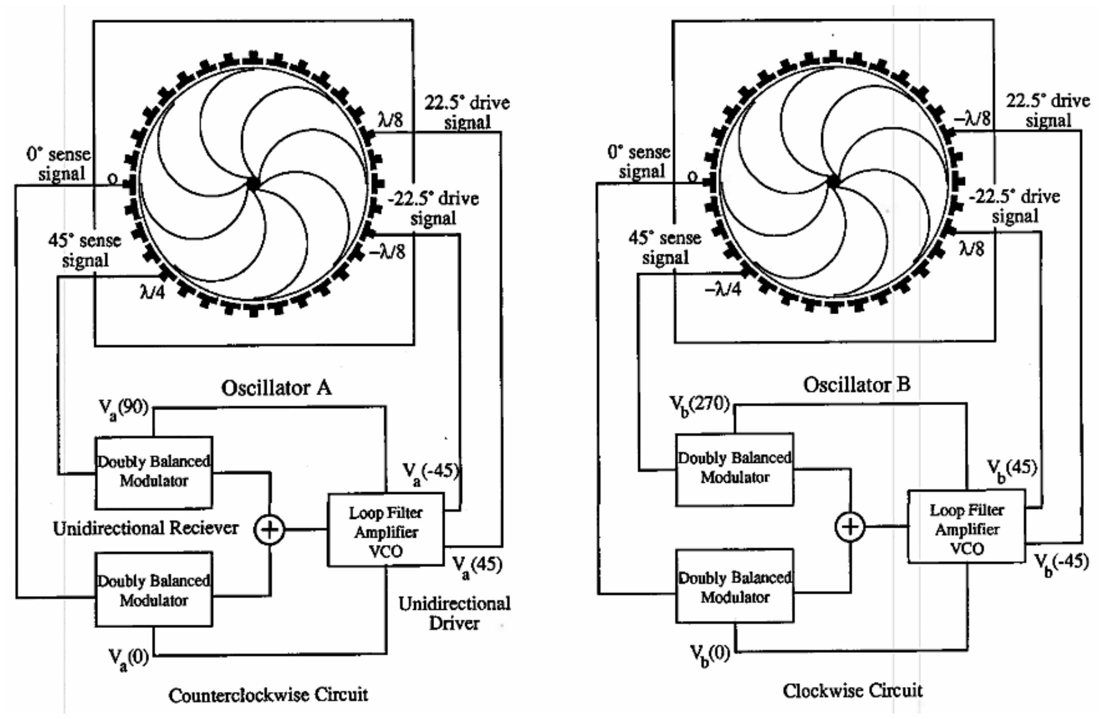


Figure 1.6: Diagram of PLL based sustaining control for RIG. One alternative control, from [14], for a RIG is to use PLLs to generate the correct control signals for a RIG.

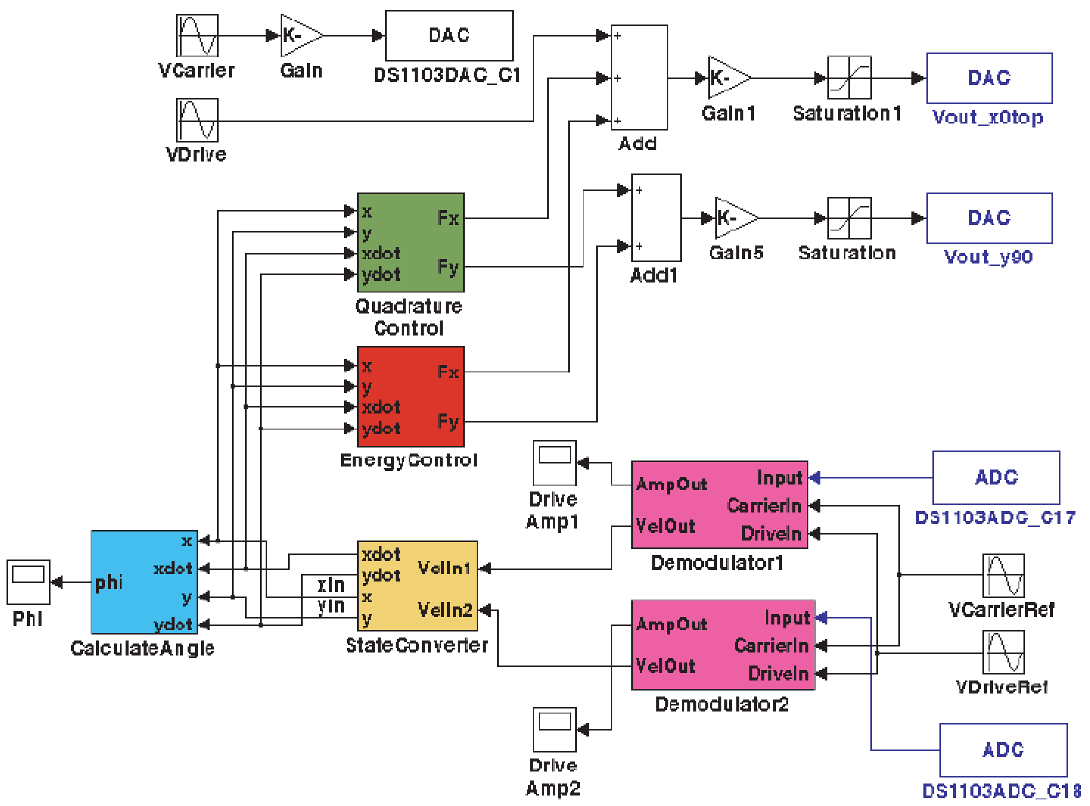


Figure 1.7: Diagram of control with quadrature compensation in SIMULINK. Software based control with quadrature compensation in SIMULINK was proposed in [5].

stable rate-integrating operation. One limitation of this control was that the frequency was fixed and not controlled with a PLL. Although it is not reported, it is also very difficult to make SIMULINK controls operate in real time for fast system, which may have limited the stability. Also, rather than sense the motion directly, a carrier wave was used to sense the motions. This method has been proposed previously [17] to move the sense signal to a higher frequency and help improve gain matching.

An application-specific integrated-circuit (ASIC) based system for rate gyroscope characterization and control is now commercially available [18] and for companies wishing to accelerate development of a commercial product could be an attractive option. The ASIC is programmable in the sense that the control loop and filter parameters can be changed, and it also includes an advanced calibration algorithm for temperature. The downside of the ASIC is that the control methods are fixed, or at least modifying them would require some form of intellectual property agreement. The ASIC also does not obviously allow for as flexible of characterization as the software methods presented in this thesis.

1.1.2 RIG control methods

The basic RIG controls proposed in [15] and [19] have been analyzed for their stability and effect on the angle readout in several papers such as [16]. A few groups have tried to propose controls which are substantively different from the basic controls used in earlier works. The following works are analytical and present only simulation results.

In [9], a control is analyzed where the gyroscope is driven at a fixed frequency. The forcing signals required to correct for frequency mismatch are very large for even small frequency mismatch, so the chosen frequency must be very close to the resonance. Also, it is not clear that the proposed frequency compensation loop would actually force the resonant frequencies to be equal. Damping mismatch compensation is also proposed by the means of a pre-calibrated damping ratio matrix. This method would correct the variation in the amplitude due to damping mismatch, but it is not demonstrated that this method would compensate for the drift due to damping. The simulated behavior of an ideal gyroscope and a non-ideal gyroscope with the proposed control system are show in Figure 1.8. The left half of the figure shows the non-ideal gyroscope and there are clearly still large discrepancies from ideal behavior. One significant source of noise in RIG operation is noise from the PLL used to track the resonance frequency, and so operating at a fixed frequency could help improve performance. In the controls presented in Chapter V, the gyroscope can also be operated with the PLL control loop disabled after start-up, so that the gyroscope is driven with at a fixed frequency close to the resonance. This operation has been briefly simulated with the hybrid control system and shows some promise, but it has not been extensively analyzed. One possible source of error is the effect of large angular vibrations on the resonance frequency.

In [20], a control is proposed that compensates for stiffness mismatch and combines rate and rate-integrating operation. The stiffness mismatch is measured by observing the frequency of the resonance on each axis. The measured mismatch is used to generate quadrature forces which compensate for frequency mismatch. This method assumes the frequency mismatch is aligned with the sense axes and does not provide any means to eliminate the effect of cross-axis stiffness. Damping mismatch is treated as a gain mismatch, which does not properly compensate for damping, and the error in the compensation creates more quadrature error. All of the compensation is based on measurements in a pre-run calibration phase and so are sensitive to in-run drift. Simultaneous operation in rate and rate-integrating mode is proposed to extract the angle from the residual quadrature error. Rate operation is enable by adding a modulated signal on the X-axis, away from the resonance frequency, and detecting it on the Y-axis. The rate from the rate mode and a rate observer created from the angle readout are combined to produce the final rate measurement. With this method they are able to extract the rate and angle from a simulated gyroscope with noise and uncompensated errors. The rate operation is not examined in detail, and it appears that the quality of the rate operation would depend heavily on the gyroscope orientation. This style of dual-mode operation is an interesting candidate for implementation using the control system in Section 5.6.

1.2 Review of characterization and tuning methods

The frequency separation of gyroscope modes determines the gain and open-loop bandwidth of a vibratory gyroscope. To improve the performance of gyroscopes as

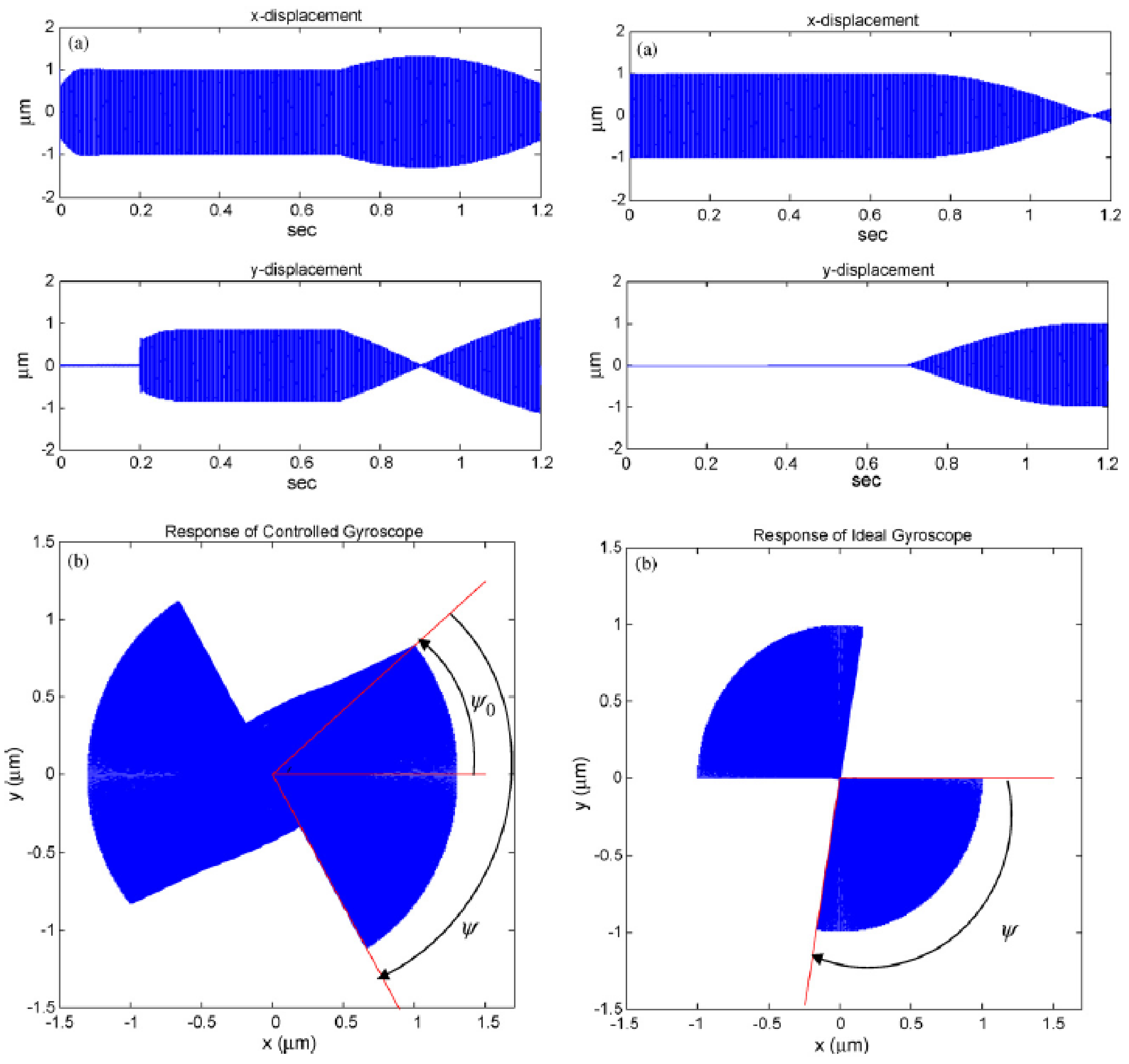


Figure 1.8: Simulated behavior of a gyroscope with fixed frequency controls. From [9], the simulated behavior of a non-ideal gyroscope under their proposed controls (left plots) compared to the behavior of an ideal gyroscope (right plots). The blurring of the orientation indicates significant levels of uncompensated error.

they are reduced in size, it is desirable to match the frequencies of the resonance modes on the drive and sense axes [21]. Several online and offline methods exist to measure and tune the mode separation. The defining characteristics of each method are how the mismatch is determined, the means of adjusting the modes, and the degree of tuning that is available. Some of the more successful methods are presented below.

1.2.1 Offline Mode Matching

Offline mode matching is generally able to cope with a larger range of mismatch than online methods. Electrostatic spring softening is the most common method of tuning MEMS resonators. In [22], a method is presented to tune a gyroscope by adding mass to the gyroscope at defined points, building on earlier work by the same group on electrostatic tuning [23]. The method is presented as a means of reducing the acceleration sensitivity of the gyroscopes, but the acceleration sensitivity is directly linked to the frequency matching of the gyroscope [3]. The advantage of physically modifying the device is that tuning voltages are subject to drift and are a source of noise. The disadvantage is that the modifications generally need to do be done at fabrication time and can't be updated as the device ages.

Offline electrostatic tuning methods for ring gyroscopes are presented in [24] and [25]. Electrostatic tuning uses the electrostatic spring softening effect to change the effective stiffness. Either paper could have formed the basis of the automatic tuning presented in Section 3.2. In [25], the tuning of a ring gyroscope is broken down into a cross coupling (g) and on-axis tuning ($\omega_1 - \omega_2$), illustrated in Figure 1.9. Perfect tuning is possible only when both the cross-coupling and on-axis mismatch terms are eliminated. A measurement scheme and method of calculating the tuning values required for matching a ring gyro are provided, but the experimental results are limited because the gyroscope used only had the capability of adjusting the on-axis

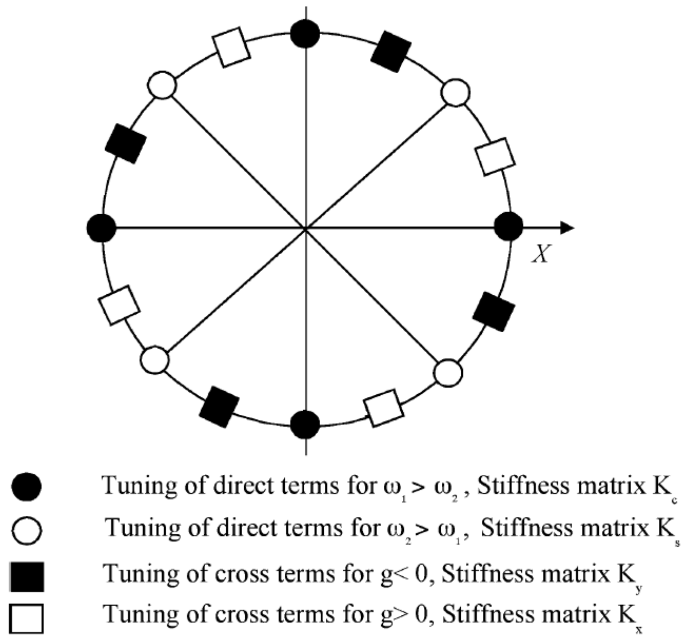

X
● Tuning of direct terms for ω1 > ω2 , Stiffness matrix Kc
○ Tuning of direct terms for ω2 > ω1, Stiffness matrix Ks
■ Tuning of cross terms for g< 0, Stiffness matrix Ky
□ Tuning of cross terms for g> 0, Stiffness matrix Kx

Figure 1.9: Classification of tuning electrodes for cross-coupling and on-axis tuning. From [25].

frequency terms. The best tuning achieved was approximately 10 Hz from an initial mismatch of approximately 250 Hz, Figure 1.10. The projected tuning voltage was 59.4±4.9 V and the measured optimum voltage was 58 V. One apparent advantage of this method for calculating the mismatch is that one only need to drive and sense one axis to measure the mismatch parameters and compare the phase difference and ratio of the response at the two modes. For the phase, the expected values are 0 or pi, and so small phase errors are not significant. However, damping error will cause a large error in the measured frequency mismatch orientation because the peaks on one axis may be very different for each mode due to damping mismatch, but the ratios of the response between the two axes for each mode is relatively insensitive to damping mismatch. Also, when the gyroscope cross-coupling is tuned to be very small, it would be very difficult to determine the residual mismatch using the response from only one axis since the response of one mode will be very small, possibly even below the noise floor.

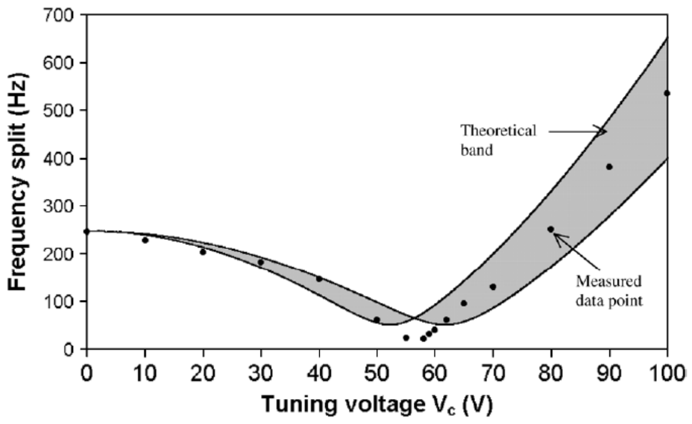


Figure 1.10: Tuning of a gyroscope without cross-coupling removed. Since electrodes for tuning the cross coupling were not available in [25], the optimal frequency mismatch is still 10Hz.

A algorithm for electrostatic frequency tuning was presented in [23] which could be automated, but automated operation was not demonstrated. The method is based on fitting a number of parameters in a gyroscope model to the measured gyroscope response using standard optimization techniques. The gyroscope response is measured by driving one axis and sensing each axis and then repeating for the other axis, for four measurements per step. The fitting method returns quality factors and frequency mismatch results, Figure 1.11 that could not be derived from linear interpolation between the measurement points. Being able to measure the gyroscope response using lower resolution tests would significantly decrease the time required for automatic tuning. The fit of the low resolution data is limited when the response is closely tuned, Figure 1.12. The low resolution response indicates that the gyroscope is perfectly matched, while the high resolution response indicates a 15 mHz mismatch, which is still very good. Part of the mismatch could be due to drive amplitude effects if different measurement methods were used to obtain the high and low resolution response. The initial separation was only 4 Hz with this gyroscope and some adjustment would have to be made for gyroscopes with a large initial mode separation, where the mechanical response between the modes is below the noise floor. There are

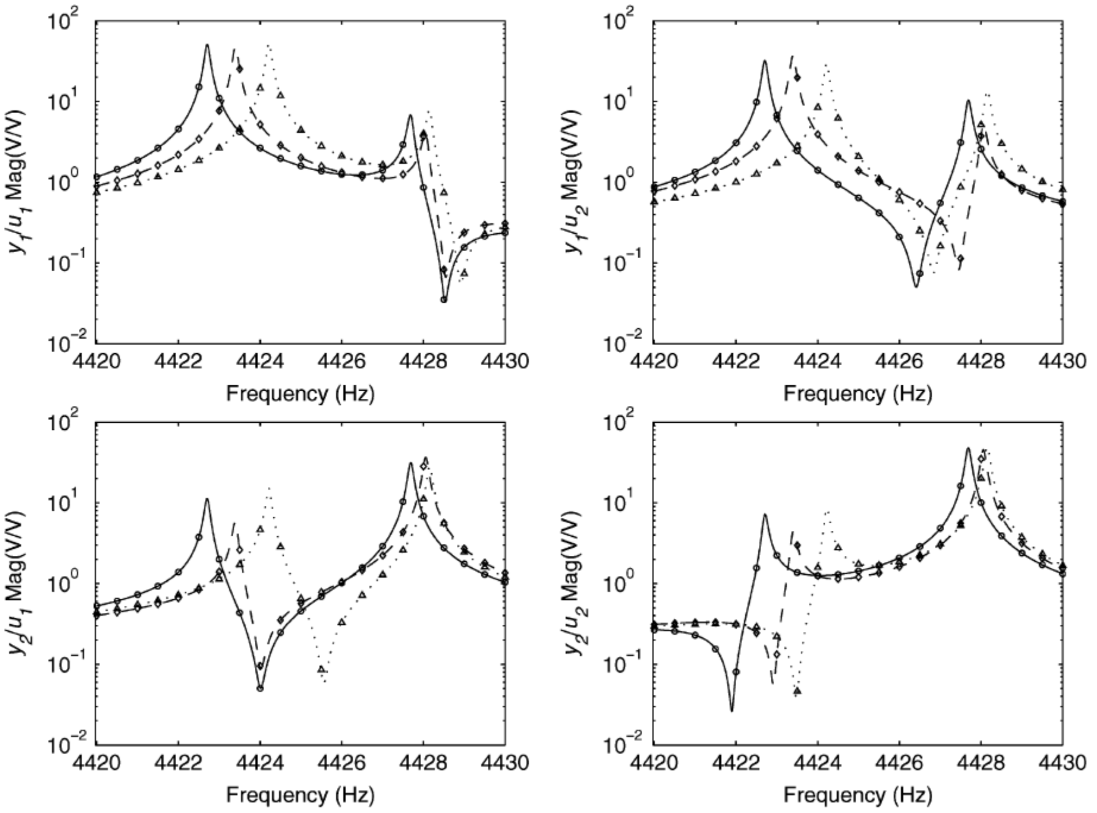


Figure 1.11: Resonator response using optimized model fit. In [23], the gyroscope response is determined by fitting measurements to a model. The lines are the fitted response with different tuning voltages and the markers are the measurement points. The quality factor and frequency resolution are higher than would be measured from a linear interpolation of the measurement points. The tests are used to calibrate the gyroscope response to the tuning voltages.

no significant barriers to implementing this method in the proposed characterization system in Section 3.1, which speaks to the flexibility of the system. In fact, these tests could performed much more quickly using the dual axis drive and sense functionality of the system presented in this thesis.

Closed-loop mode-matching was proposed in [12] using a FPGA based system which is also used for closed-loop gyroscope control [7]. The proposed system can switch which sense axis is the drive and which is the rate sense axis. The mode matching method was to measure the resonance frequency with the one axis set as

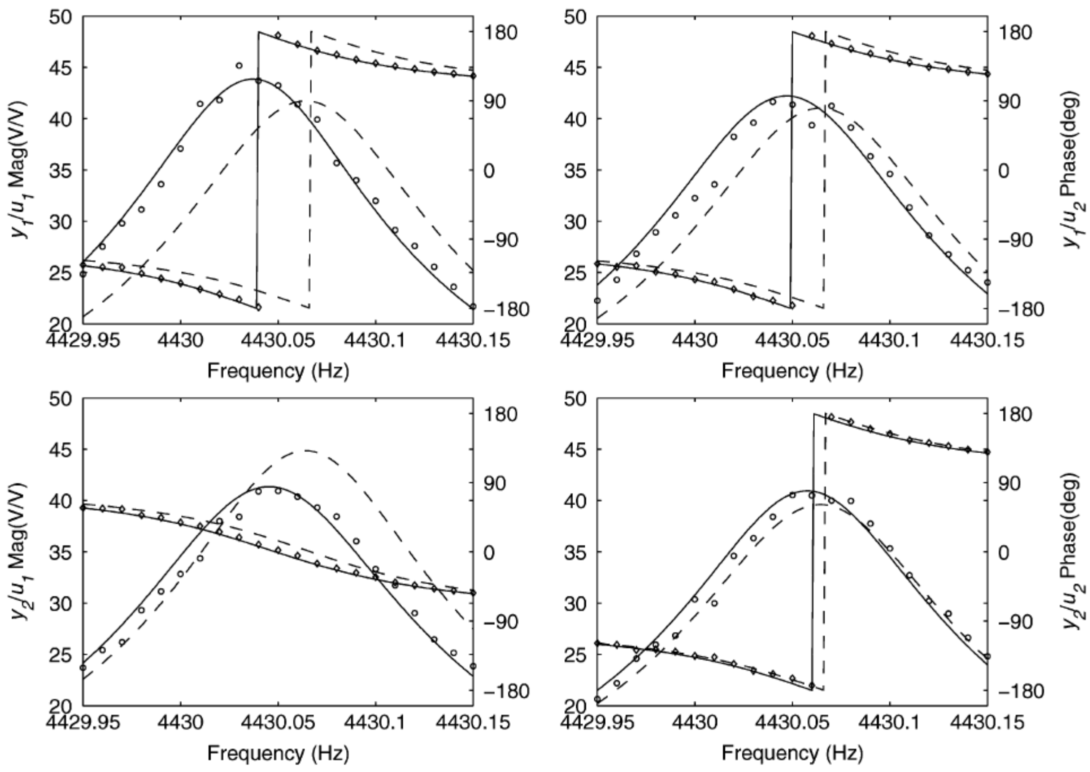


Figure 1.12: Resonator response using optimized model fit compared to high resolution scan. From [23], the fit of data measured using a lower resolution (0.5 Hz) does not match the result measured with a higher resolution (10 mHz). The 10 mHz magnitude and phase are indicated with markers, the dashed lines are the the fit from 0.5 Hz data and the solid fit is a fit to the 10 mHz data.

the drive axis, and then repeat the measurement with the other axis. Each measurement would take about 1 second assuming good signal-to-noise ratio (SNR) and could accurately determine the resonance frequency. No method was proposed for automating the tuning however. The quality of the matching was proposed to be determined by the level of zero-rate offset, which they determined to be at a minimum with 0.1 Hz mismatch. No results were given for the level of matching they were able to obtain or how much improvement was found. The same group is working on automatic control algorithms, however they claim the hardware would need to be supplemented with a microprocessor to implement the automatic control.

1.2.2 On-line tuning

Several on-line tuning algorithms rely on the phase shift between drive and sense modes for tuning [12, 21, 26]. The advantage of this approach is that the phase shift is easily detected and the tuning can be performed quickly. These methods generally rely on the stiffness matrix not having any off-axis components so that a single tuning voltage can match the modes. This assumption is not generally valid however and will limit the degree to which the modes can be matched.

In [26], the mode-matching scheme based on measuring the difference in phase as in Figure 1.13 is proposed. When the gyroscope is tuned, the phase response of the two modes will be equal. The mode matching control consists of a phase error detector and a PI controller, the output of which is applied to a tuning electrode. The initial phase difference of a gyroscope with roughly 100 Hz mismatch is tuned from a 90° phase shift to 1.3°and the rate sensitivity is increased by a factor of 21. The gyroscope frequency is around 9 kHz and the quality factor is specified as 800, so this improvement is smaller than expected. Mismatch in the phase response of the sense amplifiers for the drive axis and rate axis would crate an offset that would limit the matching accuracy. A significant limitation of this approach is that it requires

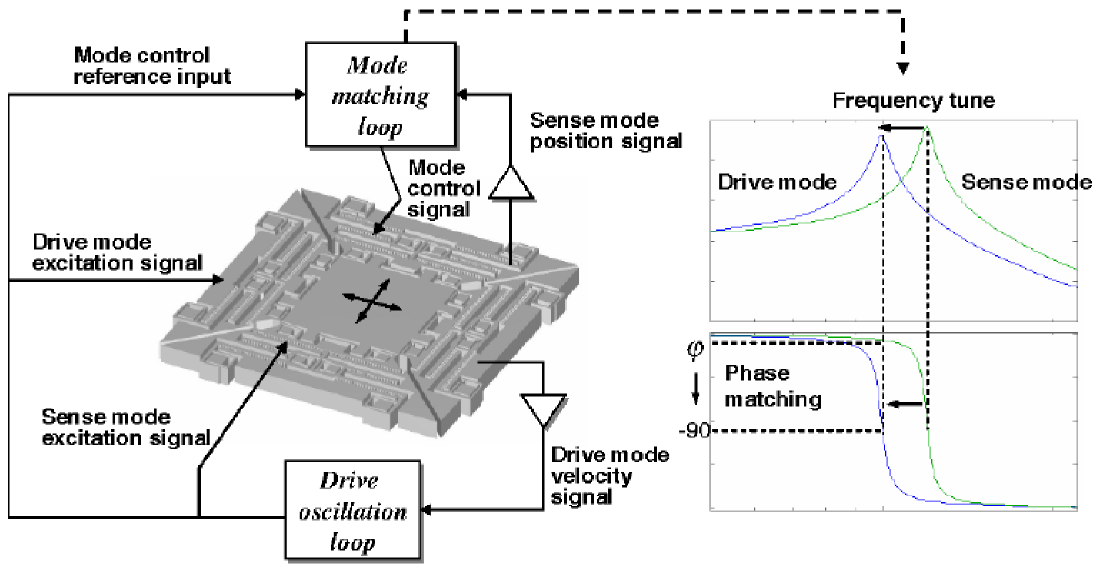

Mode control reference input
Mode matching loop
Sense mode position signal
Mode control signal
Drive mode excitation signal
Sense mode excitation signal
Drive oscillation loop
Drive mode velocity signal
Frequency tune
Drive mode
Sense mode
φ
Phase matching
-90

Figure 1.13: Closed-loop tuning using phase-matching. From [26], mode matching using the phase error between the drive and sense channels was proposed and tested for a rate gyroscope.

a signal on the rate axis which would not be there if the gyroscope were completely matched and zero rate applied.

An offset compensation method for non-mode matched gyroscopes is proposed in [27] which attempts to separate the signal due to a Coriolis Force from signals due to electrical feed-through and drive axis misalignment by modulating the drive signal in a way that should be outside the response rate of the high-Q resonator. This compensation is not meant to address mode-matching, but the same principle could easily be extended to mode matching. The modulation can be achieved either by directly modulating the drive signal or by applying an AC signal to the compensation electrodes. The offset components are determined by demodulating the received signals with the modulation signal. Offset compensation is an active area of research currently, but other methods such as [4] and [28] disrupt normal operation. The primary challenge of the method proposed by [27] is applying a signal that is close enough to the gyroscopes sensitive range that the effect can be measured accurately without interfering with sensing of the Coriolis acceleration.

This method of compensation could be tested with the control system proposed in Section 5.6 by modulating and demodulating the drive and sense signals. Compensation techniques using DC electrodes could then be tested by manually tuning the DC values and observing the effect on the extracted offset components. Automation could be accomplished using remote control of DC power supplies or using the auxiliary DAC outputs to control the electrode biases directly.

1.3 Thesis Contributions and Organization

This work focuses on system level contributions to the characterization and control of MEMS gyroscopes. The proposed contributions are

- A characterization and control system for MEMS gyroscopes built on open and commercially available hardware and software. The characterization tools are much more complete than other work and include ring-down analysis, impulse response, and gain-phase analysis. The system is easy to use and expand so that other researchers can test their own devices or algorithms without large investment in hardware or time.
- Automatic mode-matching of high-Q (Q¿50,000) MEMS gyroscopes from over 40 Hz mismatch to below 100 mHz.
- An efficient FPGA implementation of rate gyroscope control with amplitude, rate and quadrature closed-loop control demonstrating better than 400% improvement in performance for a MEMS gyroscope over open-loop control.
- Hybrid software and firmware control system and gyroscope simulator for development of advanced gyroscope control algorithms and experimental testing of those algorithms. The hybrid architecture allows delay sensitive components to be implemented in the FPGA while other controls can be implemented in

software where development is easier.

- Rate-integrating MEMS gyroscope operation for much much longer than the ring-down time as well as and characterization of the anisotropy, which is critical to development of improved models.

- New control algorithms for rate-integrating gyroscopes to dynamically measure damping and frequency anisotropy and compensate angle drift. Compensating angle drift allows rate-integrating operation at lower rates so that dual-mode MEMS gyroscopes are practical.

The thesis is organized as follows. Chapter II presents the theory of MEMS vibratory gyroscopes and the interface and control hardware used in this work. Chapter III presents a system for characterization and and tuning of MEMS vibratory gyroscopes. Chapter IV discusses control of rate gyroscopes and presents a programmable FPGA based system for rate gyroscope control. Chapter V discusses the operation of RIGs and presents a programmable hybrid firmware/software system for rate and rate-integrating gyroscope control. Chapter VI presents advanced controls referred to as dynamic mismatch compensation to improve the performance of a MEMS RIG. Finally Chapter VII summarizes the contributions of this work, and outlines some future work.

CHAPTER II

Inertial Sensing

Inertial sensors measure the forces acting on a mass. The two main categories are accelerometers for sensing acceleration and gyroscopes for sensing rotation. Vibratory gyroscopes detect rotation through the Coriolis force (or Coriolis acceleration), Equation 2.1, an apparent force experienced by a moving object in a rotating reference frame. In this work we focus on resonant MEMS gyroscopes. This chapter covers the Coriolis force, gyroscope mechanics, and the electronics used to interface with the gyroscope.

$$F_c = 2m\Omega\dot{u} \tag{2.1}$$

where m is the mass and $\dot{u}$ is the velocity perpendicular to the axis of rotation.

The Coriolis force can be understood using the example of a ball thrown on a spinning merry-go-round as in Figure 2.1, based on the discussion in [29]. For a ball thrown on a merry-go-round in a vacuum without gravity, from the point of view of an observer floating over the merry-go-round, the ball travels in a straight line because there are no forces acting on it. For an observer standing on the merry-go-round however, the ball appears to curve in the opposite direction from how the merry-go-round is spinning. The acceleration of this curve is the Coriolis acceleration, and the force required to create the apparent acceleration is the Coriolis force.

To make the ball travel in a straight line from the point of view of the observer

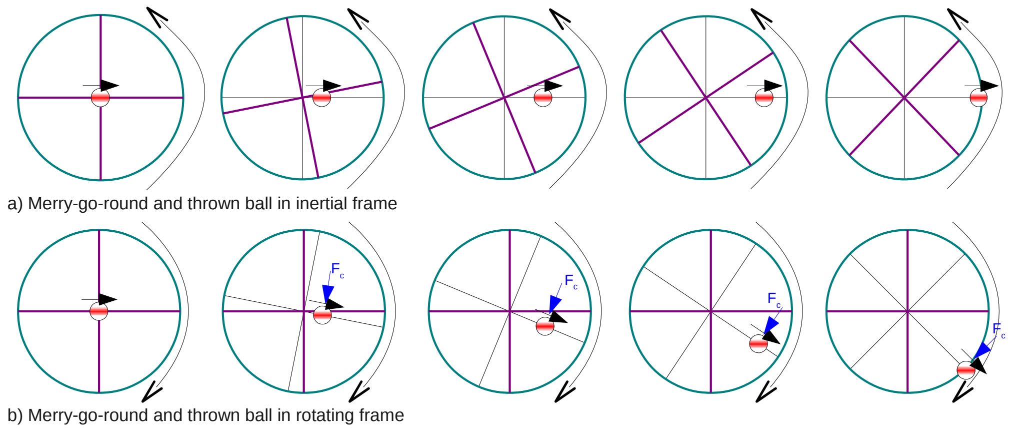

a) Merry-go-round and thrown ball in inertial frame
b) Merry-go-round and thrown ball in rotating frame
F_c
F_c
F_c
F_c

Figure 2.1: Diagram of forces acting on a moving object in rotating frame. An object in a rotating frame is affected by the centripetal force, angular acceleration, and the Coriolis Force

on the merry-go-round, a force would need to be applied with the opposite direction and magnitude as the Coriolis force, and then the observer over the merry-go-round would see the ball curve. Unlike centripetal acceleration and rotational acceleration, the Coriolis acceleration is proportional to the velocity of the ball and not the position.

2.1 Gyroscope Model

A simple conceptual vibratory gyroscope is illustrated in Figure 2.2. The motion of the mass can be described by writing equations for the motion along the X and Y axes, Equation 2.2, which is Equation 6 in [19]

$$\begin{aligned}&\ddot{x} - 2A_g\Omega\dot{y} - 2A_g\dot{\Omega}y + \frac{2}{\tau}\dot{x} + \Delta\left(\frac{1}{\tau}\right)(\dot{x}\cos 2\theta_\tau + \dot{y}\sin 2\theta_\tau)\\ &\quad + \left(\omega^2 - c_r\Omega^2\right)x - \omega\Delta\omega\left(x\cos 2\theta_\omega + y\sin 2\theta_\omega\right) = f_x\end{aligned} \tag{2.2a}$$

$$\begin{aligned}&\ddot{y} + 2A_g\Omega\dot{x} + 2A_g\dot{\Omega}x + \frac{2}{\tau}\dot{y} + \Delta\left(\frac{1}{\tau}\right)(-\dot{x}\sin 2\theta_\tau + \dot{y}\cos 2\theta_\tau)\\ &\quad + \left(\omega^2 - c_r\Omega^2\right)y - \omega\Delta\omega\left(-x\sin 2\theta_\omega + y\cos 2\theta_\omega\right) = f_y\end{aligned} \tag{2.2b}$$

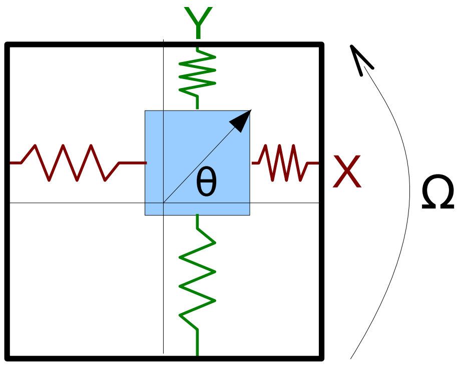


Figure 2.2: Single mass Coriolis vibratory gyroscope. A single mass attached to a frame with springs aligned with the X and Y axis can be used as a generic model of all Coriolis vibrator gyroscopes.

where x is the displacement on the x axis, y is the displacement on the y axis, Ω is the rotation rate, A_g is the angular gain which is a function of the mode shape, τ is the damping time constant, $\Delta\left(\frac{1}{\tau}\right)$, θ_τ is the rotation of the damping matrix from the x axis, ω is the average resonance frequency, c_r is the centripetal force scaling factor which is function of the mode shape, $\omega\Delta\omega$ is the frequency mismatch, θ_ω is the rotation of the frequency matrix from the x axis, and f_x and f_y are the external forces applied by drive electrodes to the x and y axes respectively. The damping mismatch and frequency mismatch are defined as

$$\Delta\left(\frac{1}{\tau}\right) = \frac{1}{\tau_1} - \frac{1}{\tau_2}$$
$$\omega\Delta\omega = \frac{\omega_1^2 - \omega_2^2}{2}$$

where τ_1, τ_2 are the damping time constants and ω_1, ω_2 are resonance frequencies when the gyroscope is aligned along the principle axes of each quantity.

The Coriolis acceleration terms $2A_g\Omega\dot{x}$, $2A_g\Omega\dot{y}$ couple the X and Y axes and allow the vibrating structure to act as a gyroscope. All of the other terms which couple the two axes are undesirable and are a source of offset in a rate gyroscope and drift in a rate-integrating gyroscope. For a rate gyroscope, the resonance frequencies are

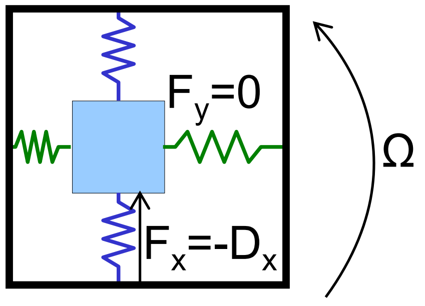


Figure 2.3: Single mass Coriolis vibratory gyroscope in rate mode. Canceling the damping on only one axis of the gyroscope causes it to operate in rate mode

not necessarily matched. Mismatch increases the bandwidth of the gyroscope. However, matching the resonance frequencies will amplify the Coriolis acceleration by the quality factor of the gyroscope, which can increase the sensitivity of the gyroscope by several orders of magnitude. The downsides of mode-matched operation are reduced bandwidth, scale factor stability and bias stability unless a control loop is used to increase the bandwidth and maintain the mode matching. Interest in mode matching of gyroscopes is increasing as a method to maintain sensitivity as gyroscopes are being made smaller to fit into commercial products and micro-aerial vehicles [30]. The gyroscopes in this thesis are typically mode-matched.

Any mode matched gyroscope can be operated in either rate or rate integrating modes. If the gyroscope is forced (F_x) to maintain an oscillation on one axis by canceling the damping force (D_x) as in Figure 2.3, while the other axis is unforced the gyroscope will operate in a rate mode. In the rate mode, the X axis is often called the drive axis or the resonator. The Y axis is the rate-sense axis or accelerometer. In the rate mode, when the gyroscope rotates the Coriolis force produces a force on the rate-sense axis proportional to the amplitude of the drive axis oscillation. This force causes an oscillation to grow on the rate-sense axis until the damping, which is proportional to the oscillation amplitude, balances the Coriolis force, as illustrated

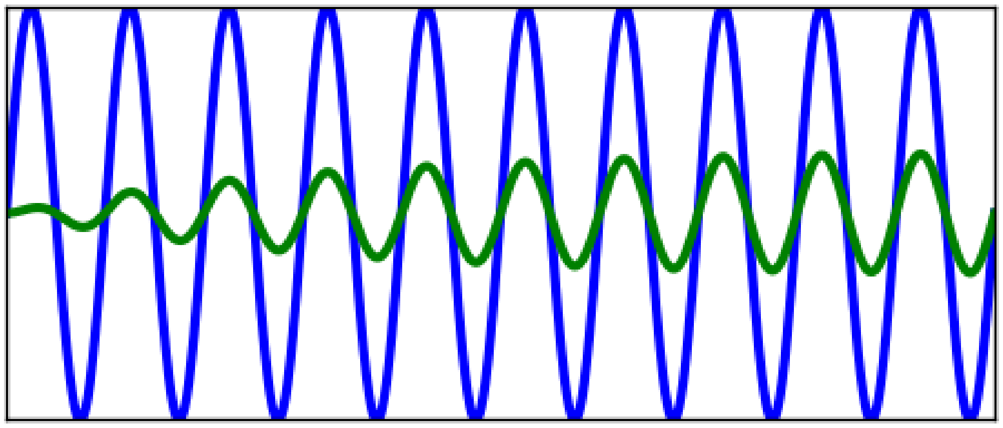


Figure 2.4: Single mass Coriolis vibratory gyroscope amplitudes in rate mode. The amplitude on the X or drive axis (blue) is maintained constant by a control circuit while the Y or rate-sense axis (green) grows until the Coriolis force is balanced by the damping.

in Figure 2.4. The output of the gyroscope is the amplitude of the rate-sense axis vibration which is

$$y = A_g \tau x \Omega. \tag{2.3}$$

It is useful to normalize this to the drive axis amplitude to get

$$\frac{y}{x} = A_g \tau \Omega. \tag{2.4}$$

Since the oscillation on the drive axis is kept constant while the Y axis amplitude changes, the total energy is not constant. The rate of the energy change is dictated by the quality factor of the gyroscope and the frequency mismatch. If the gyroscope is matched, the oscillation on the Y axis will grow at a rate equal to the damping time constant τ. The gyroscope bandwidth is then limited to the inverse of the damping time constant which typically results in a very small bandwidth.

To improve the bandwidth and stability of a rate gyroscope, the gyroscope can be operated in a closed-loop or force-feedback mode where the Coriolis force is canceled by a electronic signal to the rate-sense axis. The output signal is then the amplitude

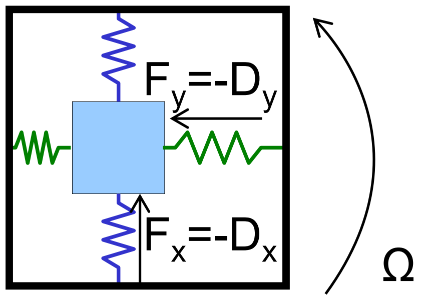


Figure 2.5: Single mass Coriolis vibratory gyroscope in rate-integrating mode. Canceling the damping on both axes of a mode-matched gyroscope causes it to operate in rate-integrating mode

of the feedback signal. This signal can be normalized against the amplitude of the drive signal to produce a rotation signal very similar to the open-loop one

$$\frac{F_y}{F_x} = A_g \tau \Omega. \tag{2.5}$$

If the gyroscope is forced on both axes in order to cancel the damping on each axis as in Figure 2.5, or if the gyroscope's damping is much smaller than the Coriolis Force, the gyroscope will operate in a rate-integrating mode. The rate-integrating mode is also known as the whole angle, direct angle, or angle mode. When the gyroscope operates in rate-integrating mode, there is no "drive" axis and "rate-sense" axis. If the gyroscope is initially oscillating along the X axis, rotation will create a Coriolis force that causes the oscillation on the y axis to grow, but without any damping the oscillation will continue to grow. Since the Coriolis force is anti-symmetric, the growing oscillation on the Y axis will damp the oscillation on the X axis causing it to decay until the original oscillation is canceled and then the X axis will start to oscillate with an oscillation in anti-phase of the original oscillation. This process will then repeat with X and Y reversed as illustrated in Figure 2.6. The output signal for a RIG is the angle of the oscillation which is derived from the ratio of the amplitudes

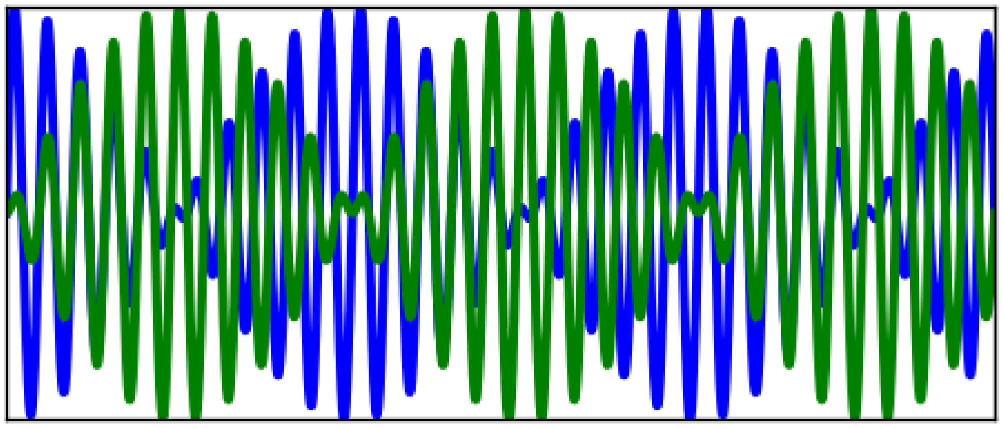


Figure 2.6: Single mass Coriolis vibratory gyroscope amplitudes in rate-integrating mode. The damping on the x (blue) and y (green) axis is canceled so that the Coriolis force is able to continuously shift energy between the X and Y axes.

on the X and Y axes

$$\frac{y}{x} = \tan \int A_g \Omega dt = \tan \theta \tag{2.6}$$

where θ is the orientation of the oscillation as illustrated in Figure 2.2. The sense signal θ is simply

$$\theta = \arctan \frac{y}{x} \tag{2.7}$$

One benefit of rate-integrating mode is that the bandwidth is ideally infinite. Physically, this is because the total energy of the gyroscope is kept constant and so the damping time constant does not mater. This infinite bandwidth can also be seen by differentiating the sense signal with a time varying rate applied such as $\Omega = A \sin \omega_\Omega t$

$$\hat{\Omega} = \frac{\theta}{dt} = \frac{\int A_g \Omega dt}{dt} = A_g \Omega = A_g A \sin \omega_\Omega t \tag{2.8}$$

where $\hat{\Omega}$ is the measured rate. Since there is no integration or differentiation, the scaling is independent of the frequency ω_Ω. Practically, for a given rotational amplitude A, as ω_Ω increases, the measured change in the angle will go to zero which implies that the bandwidth is limited by the resolution of the angle measurement.

A more significant limitation to the bandwidth is the angular acceleration term in Equation 2.2. If the rate is changing rapidly, especially if it is a step function, this term represents a force in phase with the displacement, which creates an apparent frequency mismatch and drift in the gyroscope.

2.2 Hardware

Electronics are required to turn a resonating structure into a gyroscope. These electronics can be roughly partitioned into the sense electronics and control electronics. The sense electronics are necessarily analog and are responsible for transducing the motion of the resonator into a signal which can be used as a measure of the rate or act as an input to the control electronics. This signal is generally a voltage, but could be a current or a direct digital signal such as a pulse train. The control electronics can be either analog or digital. Analog control works well for discrete implementation of relatively simple control schemes, but digital control offers many advantages when the control is to be implemented in an integrated circuit or to be made more sophisticated with advanced compensation. With digital control, it is relatively easy to share resources between multiple sensors, unlike analog control where most operation is continuous in time. There are methods for making discrete time analog controls, but the complexity would rival or surpass a digital implementation. In this work, the analog electronics used for sensing are made as simple as possible and all of the control is digital. This arrangement results in a powerful and flexible system that can be used with a wide range of sensors and control schemes.

2.2.1 Sense Electronics

Most Coriolis vibratory gyroscope (CVG) use capacitive detection schemes. Capacitive sensors work well at higher frequencies such as the resonance frequencies of MEMS devices and are generally easier to fabricate than piezo-resistive or other trans-

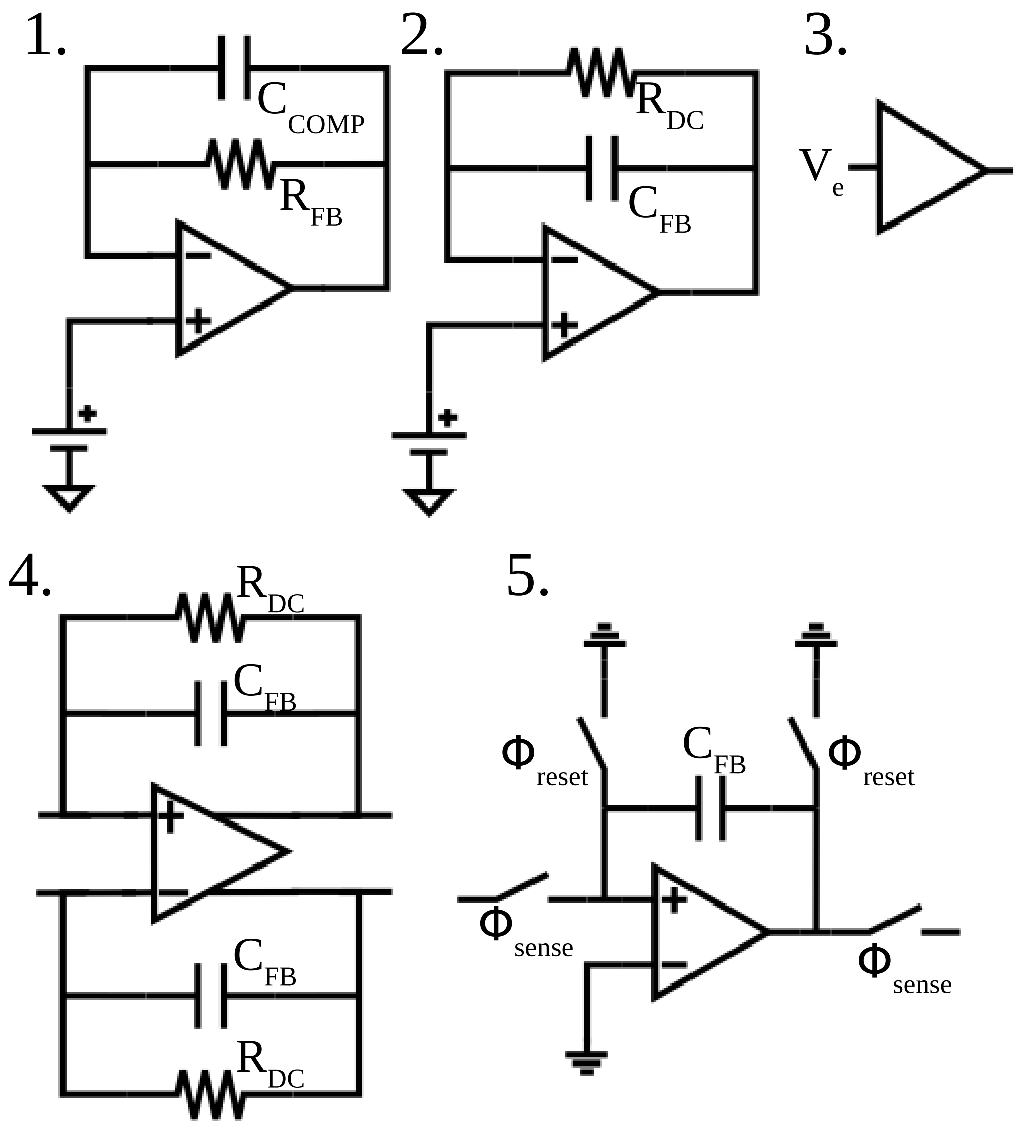


Figure 2.7: Amplifier configurations for charge sensing. 1) transimpedance amplifier, 2) charge integrator, 3) voltage amplifier 4) differential charge amplifier, 5) switched capacitor charge amplifier.

duction methods. Sense amplifiers for capacitive transducers come in many forms, the most common of which are illustrated in Figure 2.7. For a generic readout that works with a wide range of sensors, the transimpedance and charge integrator topologies are preferred. Voltage amplifiers are sensitive to the input capacitance which makes them less stable when not integrated on the same die as the gyroscope, and also have worsening performance as the gyroscope frequency increases. Switched capacitor circuits work well at low frequencies when implemented with discrete components and with proper design can have very low drift. For gyroscopes, the capacitors should be switched much faster than the gyroscope frequency which is difficult using discrete components, especially if the gyroscope frequency is near the megahertz range. Even with an integrated implementation, the switching and sampling can increase the noise [31].

Fully differential transimpedance amplifiers and charge integrators are preferable to single ended implementations, but most commercial off the shelf (COTS) fully differential amplifiers are optimized for high-speed applications and do not have good noise performance. An integrated circuit implementation customized for gyroscopes should be fully differential.

Transimpedance amplifiers and charge integrators work in very similar ways and the same amplifier can be used for either transimpedance or charge integration by changing the relative size of R_f and C_f in Figure 2.7 circuits 1 and 2. Charge integrators give better noise performance at low frequencies, but require very large resistors for R_{DC} and tend to be unstable in discrete implementations. Transimpedance amplifiers are stable, easy to implement with discrete components and work over a wide range of frequencies, with performance generally improving with the frequency. For these reasons, transimpedance amplifiers were used in this work.

The ability to detect the Coriolis force is linked to the minimum capacitance change that the sense method can resolve based on the signal to noise ratio. The

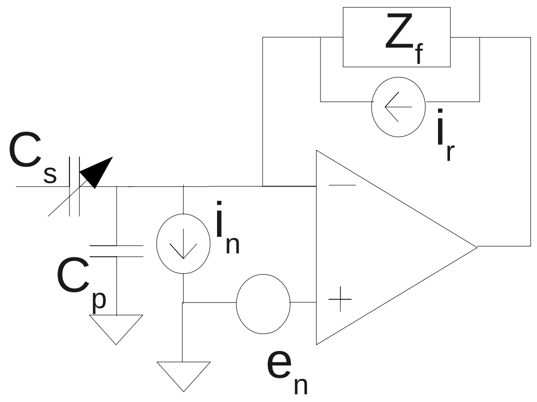


Figure 2.8: Noise model for a single amplifier with arbitrary feedback.

minimum detectable capacitance change $\Delta C_{s,min}$ for a sense amplifier is

$$\Delta C_{s,min} = \frac{I_s}{\omega V_{bias}} \tag{2.9}$$

where I_s is the input reference current noise, ω is the detection frequency, and V_{bias} is the voltage across the sense capacitor. For the readout scheme in Figure 2.8, I_s can be dominated by several difference noise sources.

If the voltage noise of the amplifier dominates, the noise depends heavily on the capacitance at the input node and the minimum detectable capacitance change can be written as

$$\Delta C_{s,min} \approx e_n \frac{C_s + C_p}{V_{bias}} \tag{2.10}$$

where e_n is the amplifier voltage noise, C_s is the sensor capacitance and C_p is the parasitic capacitance (which includes the amplifier input capacitance). If the current noise of the amplifier dominates, the following equation holds

$$\Delta C_{s,min} \approx \frac{i_n}{\omega V_{bias}} \tag{2.11}$$

where i_n is the amplifier current noise. For JFET and CMOS amplifiers, this is usually very small, under $100 fA/sqrtHz$. If Z_f is mostly resistive and can be approximated

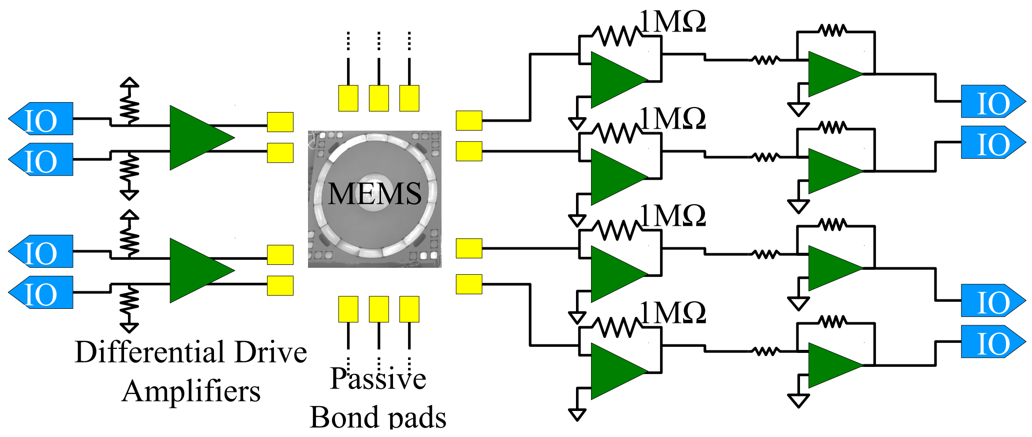


Figure 2.9: Diagram of analog interface board. A simple analog interface board was designed to mount the gyroscope and amplify the gyroscope signals

with resistance R_f, then the minimum sensed capacitance can be written as

$$\Delta C_{s,min} \approx \frac{\sqrt{4kT/R_f}}{\left[1+\omega R_f(C_s+C_p)\right](\omega V_{bias}}. \tag{2.12}$$

which simplifies to the same format as Equation 2.11 if ω is larger than the time constant, usually a few hundred Hertz, except the current noise is now the resistor current noise. This noise is typically larger than the noise of a good JFET or even CMOS amplifier and is often the dominant white-noise source. The minimum detectable capacitance change does improve with increasing R_f.

The analog interface used for most of this work is illustrated in Figure 2.9. Although differential drive amplifiers were included to provide balanced signals, they often were not used because of cross coupling problems due to the board layout. The sense amplifiers are dual op-amps in transimpedance configuration followed by buffer amplifiers to drive the signal across the slip ring on the rate-table. The transimpendance gain was usually 1 or 10 $M\Omega$ with 0.5 to 2 pF capacitors for stability. In cases where the sense capacitance was large, the differential drive and buffer gains needed to be set to unity to prevent self oscillation of higher-order modes.

2.2.2 Bias Voltage

Transimpedance and charge integrator amplifiers both require that a bias is applied across the sense capacitance. The bias voltage creates a current from the changing sense capacitance. This can be easily derived by differentiating the charge on a capacitor

$$\begin{aligned} Q &= CV \\ \dot{Q} = I &= C\dot{V} + \dot{C}V \approx \frac{dC}{dx}\frac{dx}{dt}V = \frac{dC}{dx}\dot{x}V \end{aligned} \tag{2.13}$$

where Q is the charge on the capacitor, I is the sense current, C is the sense capacitance, V is the bias voltage, x is the displacement, and it is assumed that the bias voltage is nearly constant. The sensitivity of the capacitance to the displacement is also easily derived as

$$\begin{aligned} C &= \frac{\epsilon A}{g+x} \\ \frac{dC}{dx} &= \frac{\epsilon A}{(g+x)^2} \approx \frac{\epsilon A}{g^2} = \frac{C}{g} \end{aligned} \tag{2.14}$$

ϵ is the permitivity, A is the capacitor area, g is the capacitor gap and it is assumed that the gap is much larger than the displacement x.

The bias voltage is also important for driving the gyroscope. The force due to a voltage applied to a capacitor is

$$F = CV^2. \tag{2.15}$$

There are three main methods of driving a resonator due to the dependency of the force on V^2. If $V = A_1 \sin \pi f_0 t$, that is a sinusoidal signal with amplitude A at $\frac{1}{2}$the resonant frequency, then the applied force is

$$F \propto A_1^2 \sin^2 \pi f_0 t = A_1^2(\sin 2\pi f_0 + 1) \tag{2.16}$$

which is at the resonant frequency. If $V = A_2 \sin 2\pi f_0 t$, that is a sinusoidal signal at the resonant frequency, then the applied force is

$$F \propto A_2^2 \sin^2 2\pi f_0 t = A_2^2(\cos 4\pi f_0 + 1) \tag{2.17}$$

which is a force at twice the resonant frequency. This drive method takes advantage of parametric amplification [32], which is the same as the pumping motion of swinging on a swing. If $V = V_{bias} + A_3 \sin 2\pi f_0 t$, that is a sinusoidal signal plus a DC bias voltage, then the applied force is

$$F \propto (V_{bias} + A_3 \sin^2 2\pi f_0 t)^2 = V_{bias}^2 + A_3^2 \sin 4\pi f_0 + V_{bias} A_3 \sin 2\pi f_0 \tag{2.18}$$

which has a DC component, a $2f_0$ component proportional at A^2, and a component at the resonant frequency proportional to AV_{bias}. If $A << V_{bias}$ then the component at the resonant frequency dominates and is amplified by V_{bias}.

The first two methods have the advantage of separating the drive and sense signals in frequency, which helps eliminate feed-through. The downside is that the AC voltage has to be much larger than in the third method where the AC signal is amplified by DC bias. To achieve the same force at the resonant frequency with the first and third methods requires

$$A_1^2 = V_{bias} A_2. \tag{2.19}$$

If a 10 V bias and a 100 mV drive is needed with the third method, a ≈3.2 V AC signal is required with the first method, which is 32 times larger. This means that the same signal generator can apply much larger forces with the third method and this extends the range over which the rate and quadrature controls in Sections 4.2, 5.2 can control the gyroscope.

2.2.3 Control Electronics

The central hardware for the proposed system is a USRP1 which is conceptually illustrated in Figure 2.10. The USRP1 connects to a computer via USB and has four 64MS/s 12bit ADC, four 128MS/s DAC, and an Actel Cyclone FPGA. The standard FPGA image implements decimating filters and can support up to 8MS/s across the USB bus. Daughter-boards plug in to the transmit and receive ports of the USRP1 to provide the analog interface. We have mostly used LFTX and LFRX daughter boards (also available at [33]) as the interface. The boards provide a single-ended interface to each ADC and DAC. The hardware imposed limits are summarized in Table 2.1. Delay due to the USB bus and buffering is the most significant limitation, especially for control.

The limitations of the USRP are also important to consider when designing the analog front-end to maximize performance. The ADC on the USRP is 12 bits, but this signal is decimated and filtered by at least a factor of 16 giving 14 effective bits [44], and for this work a decimation of 128 is used, giving 16 effective bits inside the FPGA. The full scale range of the USRP is $2V_{pp}$, so the least significant bit (LSB) with 16 bits is $\approx 61\mu V$. The noise of the amplifiers in the USRP are lower than this and a quiet USRP will simply read zero. To avoid this digitization limiting performance, the signal should be amplified so that the sense amplifier noise or mechanical noise is close to the LSB. With a 1 MΩ transipedance amplifier, the sense amplifier noise is generally dominated by the resistor and is $\approx 0.13\mu V/\sqrt{Hz}$ leading to an optimal gain of around 500. The USRP has a variable gain amplifier that can provide up to 20 dBV (10x) of gain, so the external electronics should have a gain of 50. In the test setup used for this experiment however, the rate chamber also adds significant noise and large voltage spikes. To reduce the effect of these, the gyroscope was driven so that the output signal was close to the limit of $2V_{pp}$ with an external gain of only 10. The USRP may be increasing the ARW by a factor of five for a gyroscope dominated

Table 2.1: USRP Hardware Summary

Sense ADC	12b, 64MS/s	Drive DAC	16b, 128MS/s
LFRX Range	$\pm 1V$, 30MHz	LFTX Drive	$\pm 3V$
Bandwidth	4MHz	Software Delay	10ms±2ms

Table 2.2: Gyroscope limitations due to USRP hardware.

Modes	PLL Loop Delay	Control Delay	Logic Resources	Max f_0	Min $Q/\pi f$
FPGA	2 μs	2 μs 12,000 LUT	2 MHz	20 μs	
Hybrid	2 μs	10 ms	Ũnlimited	2 MHz	100 ms

by electrical noise.

The control system is agnostic of the sense amplifiers and detection method except for the phase shift introduced by the sense amplifiers and whether the sense signals are in-phase with the gyroscope vibration displacement or velocity. Voltage sensing methods such as in [24] or charge amplifiers [10] produce an output signal in-phase with the vibration displacement. Current sensing mechanisms as used in [34] create an output signal in-phase with the vibration velocity. To account for this difference, it is only necessary to change the sign of the gains in the various control loops. The analog interface used for the experimental results presented here is illustrated in Figure 2.9. The circuits consist of differential drive amplifiers, transimpedance amplifiers (TIA) to convert the current of the varying capacitor to a voltage, and a second set of amplifiers to boost the signal. Differential sensing is very important to reduce feed-through of the drive signals due to parasitic capacitance, reject common-mode noise introduced by the rate-table slip rings and reject 60 Hz noise and harmonics. The TIA amplifier sensing method is used because it provides a virtual ground making it insensitive to parasitic capacitance at the sense nodes.

The choice of control electronics limits the selection of gyroscopes which can be controlled with the systems presented in Section 4.2 and Section 5.6. Table 2.2 summarizes the gyroscope resonator requirements due to the sampling rates and delays of the controls when implemented in the FPGA or in software.

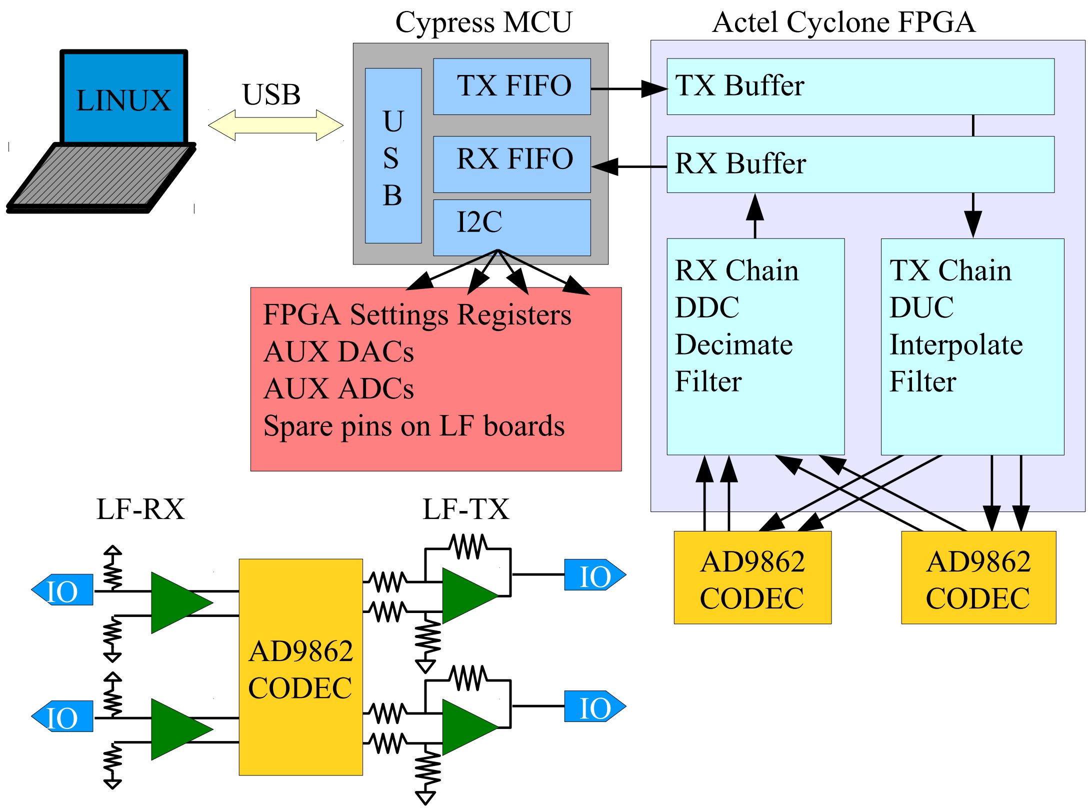


Figure 2.10: Diagram of software defined radio hardware. The USRP1 from Ettus Research was used for some digital signal processing and to link the analog interface to the computer.

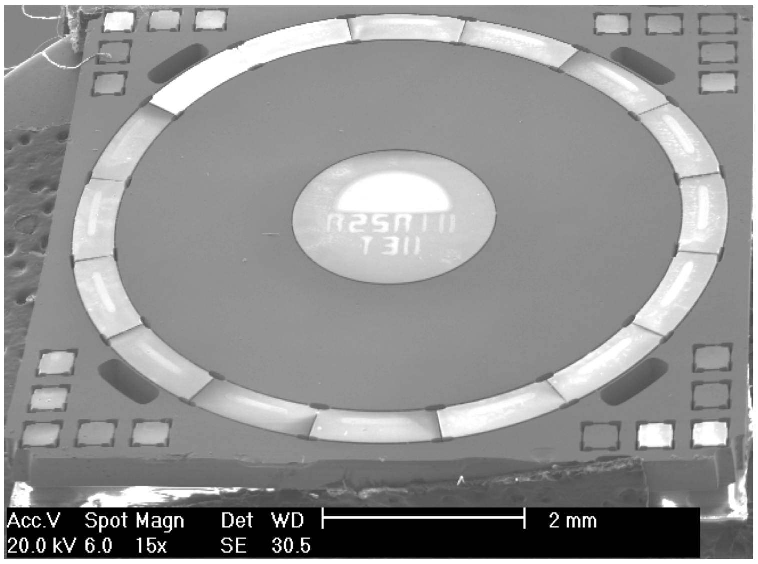


Figure 2.11: SEM of a CING gyroscope. A CING gyroscope was used for testing the rate control system. The CING is an axisymetric gyroscope with 3 kHz and 18 kHz versions with a quality factor of several tens of thousands.

2.3 CING gyroscope

Most testing of the control system was performed with a single-crystal-silicon Cylindrical Rate-Integrating Gyroscope (CING) gyroscope. The CING is fabricated using an silicon on glass (SOG) process which is described in detail in [34] and [35]. The CING is made of a silicon wafer aligned on the (111) plane and operates at the fundamental flexural (wineglass) mode. There are two versions of the CING which operate at 18kHz (radius: 2.5 mm, height:300 μm, Q≈60,000) and 3 kHz (radius:6 mm, height:300 μm, Q≈100,000) respectively. Their resonance frequencies can be electronically matched within a fraction of their 3dB bandwidths.

Although the performance of the CING gyro suffers because of a small angular gain, the large quality factors help balance this effect to create a moderate performance gyroscope.

2.4 Error Parameters

The simplest way to write the output signal of a rate gyroscope is

$$\hat{\Omega} = S_f(\Omega + \Omega_0) \tag{2.20}$$

where $\hat{\Omega}$ is the output signal, S_f is the scale factor, Ω is the applied rate and Ω_0 is the Zero-rate offset (ZRO). The ZRO is the measured rate when no rate is applied and, if it is constant, can be removed by simple calibration. Typically the ZRO is relatively stable while the gyroscope is powered on, but has large variations between power-ups. The noise in the measurement signal, stability of the scale factor, and stability of the ZRO are all very important to the gyroscope performance and contribute to different components of the noise spectrum. The noise model used here follows a standard developed for rate-laser gyroscope (RLG) [36], but is commonly used for MEMS gyroscopes as well.

Gyroscope noise is commonly broken down as quantization or rate environment noise, ARW, bias stability, and RRW. The different noise types are characterized by frequency dependency, which affects the behavior of the noise as the signal is integrated or averaged. Correlated noise and interfering signals are also important error sources. ARW is white noise in the rate which when integrated creates uncertainty in the angle which increases with the square-root of the integration time. The noise is usually given in units of $^\circ/s/\sqrt{Hz}$ or $^\circ/\sqrt{Hr}$. This noise is most important over short time spans, usually a few seconds or minutes in MEMS gyroscopes, although it can be several hours in high-end inertial grade gyroscopes [4]. Bias stability is due to $1/f$ noise in the electronics and short term variation in the ZRO. When integrated, it creates uncertainty in the angle which grows linearly with time. The units are typically $^\circ/s$ or $^\circ/Hr$. Stability of MEMS gyroscopes is frequently poor and so bias stability is tracked as the primary metric for how MEMS gyroscopes are improving

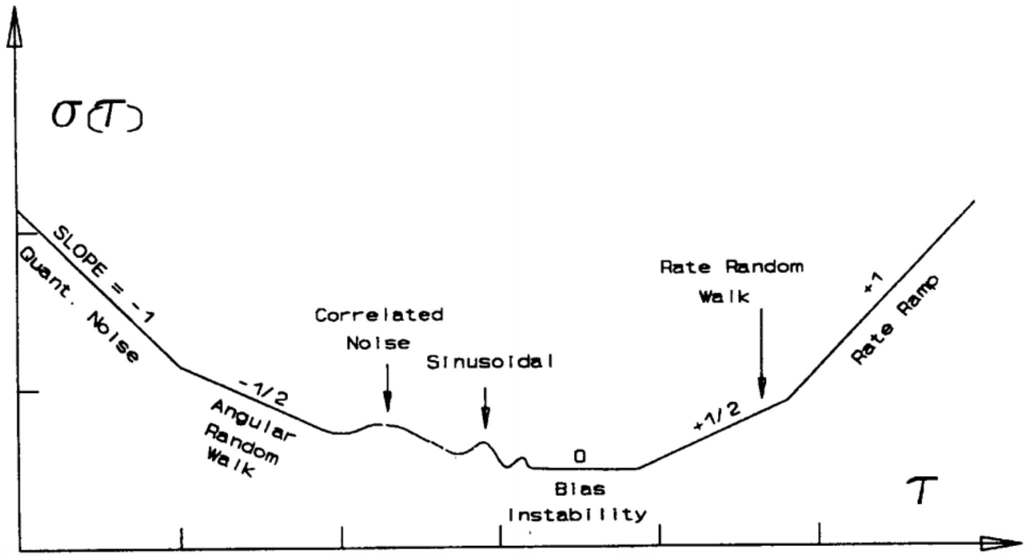


Figure 2.12: Allan variance plot showing the different noise types and their slopes when plotted as a log-log plot. From [36], The horizontal axis is the averaging time and the vertical axis is the standard deviation of the rate signal when averaged for the averaging time.

with time. RRW is characteristic of long-term drift in the ZRO. This can be due to changes in the mechanical stress shifting the offset, charging of the bias voltage capacitance, temperature drift, or other factors. RRW creates uncertainty in the measured rotation angle which grows with time squared and for high many gyroscopes it is what actually limits long term performance.

The Allan variance plot is one method of visualizing and extracting these various error sources [36], which is illustrated in Figure 2.12. Each noise source is identified by a different section of the Allan variance plot with a characteristic slope. The ARW slope is $-1/2$ and the bias stability is the flat section.

Allan variance is provided by a number of statistical software tools. It is calculated according to the formula

$$\sigma(\tau)^2 = \frac{1}{2N}\sum_{k=1}^{N}(\bar{\Omega}_{k+1} - \bar{\Omega}_k)^2 \tag{2.21}$$

where τ is the averaging time, $\bar{\Omega}$ indicates the average over time τ so that $\bar{\Omega_k}$ is the average rate over the kth time period τ, and N is the number of periods of duration τ in the data set to be analyzed. Normally, τ for which $N < 9$ are discarded. The Allan Variance used in this work is calculate using Python and scipy [37] using the function

```
from scipy import array, unique, zeros, mean, log10, logspace
def avar(omega, fq, pts=1000, minbin=9):
  ''' omega is the rate data, fq is the sample rate, pts is the
      maximum number of Allan Variance points to calculate and
      minbin is the minimum number of data points for each Allan
      Variance point. '''
  taus = logspace(0, log10(len(omega)/minbin),pts)
  taus = array(unique([int(t) for t in taus]))
  av = zeros(len(taus))
  for it,tau in enumerate(taus):
    nbins = len(omega)/tau
    bins = omega[:nbins*tau] # ensure all bins are equal size
    bins.shape = (nbins, tau)
    bar_omega = bins.mean(axis=1)
    av[it] = mean((bar_omega[1:]-bar_omega[:1])**2)/2
return taus/fq, av
```

The Allan variance description of noise is not the only way of considering gyroscope noise and there are a number of proposed replacements and augmentations of Allan Variance [38–40], but the Allan variance method is still the standard. For rate-integrating gyroscopes, noise analysis is less well defined and some methods of translating rate-integrating performance into effective rate noises are presented in Section 5.4.

If the modes of a gyroscope are matched, the Coriolis force as at the resonance frequency of the gyroscope rate-sense axis and in phase with the velocity, which means the displacement on the drive axis and the displacement on the rate-sense axis due to the Coriolis Force are in phase. If the gyroscope is not mode matched, the rate-sense axis is driven off-resonance leading to a $\approx 90°$ phase shift between the drive and rate-sense axis displacements. Quadrature error is the component of the displacement on the rate-sense axis that is out of phase with the displacement caused by the rotation rate. This means that for mode-matched gyroscopes, the quadrature error is out-of-phase with the drive signal and for non-mode-matched gyroscopes the quadrature error is in-phase with the drive signal. Quadrature error in a mode-matched gyroscope indicates that the modes are not perfectly matched and so mode-matched gyroscope control systems often use quadrature error in a control loop to tune the gyroscope to remove frequency mismatch.

Bandwidth in a mode-matched gyroscope can also be looked at as the effective mismatch of the gyroscope due to the time varying rate. The AC component of the rate modulates the Coriolis force and shifts it away from the drive axis frequency. This is turn causes the rate-sense axis to be driven off resonance. The 3dB bandwidth of the gyroscope rate response is therefore equal to the 3dB bandwidth of the rate-sense axis resonance peak. One way to improve the bandwidth is to mismatch the modes so that the Coriolis force is applied to a less steep part of the rate-sense axis frequency response curve. Another method is to create a gyroscope with lower damping on the rate-sense axis. However, both of these methods reduce the sensitivity of the gyroscope.

2.5 Summary

In this chapter, the principles of CVG were presented as well as the hardware platform, electronics, and specific MEMS gyroscope used in the rest of this work. The electronics were chosen for their compatibility with a wide range of devices and relative insensitivity to parasitic capacitance. Although lower noise electronics are possible, the purpose of this work is to explore the system level aspects of gyroscope performance. The parameters used to describe gyroscope performance such as ARW, bias stability, bandwidth, and quadrature error were also presented.

CHAPTER III

Characterization and Compensation

We propose a system for resonant inertial sensor characterization and control using commercial SDR hardware (a USRP1) [33] as the analog/digital interface and a mixture of software and firmware for digital signal processing and sensor control. This system allows labs to evaluate their devices without onerous investment in test equipment, provides a more integrated environment for sensor test and control, and provides a framework for the development of new gyroscope control algorithms. The integration of multiple test functions and multiple test channels in a single system greatly reduces test time and errors due to re-cabling or not having the right equipment. The system is not intended to be a better network analyzer or real-time spectrum analyzer than expensive dedicated hardware, but to perform all of the functions required for high-Q, low-frequency resonator characterization in one inexpensive, flexible, and easy to use package. Performance, and cost, could be increased by moving the system to a platform with a dedicated digital signal processor, or the algorithms developed here could be moved to an ASIC for inclusion in commercial products. The resonant devices used in this work are MEMS gyroscopes, yet the applications extend to resonant accelerometers and other resonant sensors.

The control and characterization software is built upon the open source GnuRadio [41] framework and offers a powerful user interface. For characterization, we have

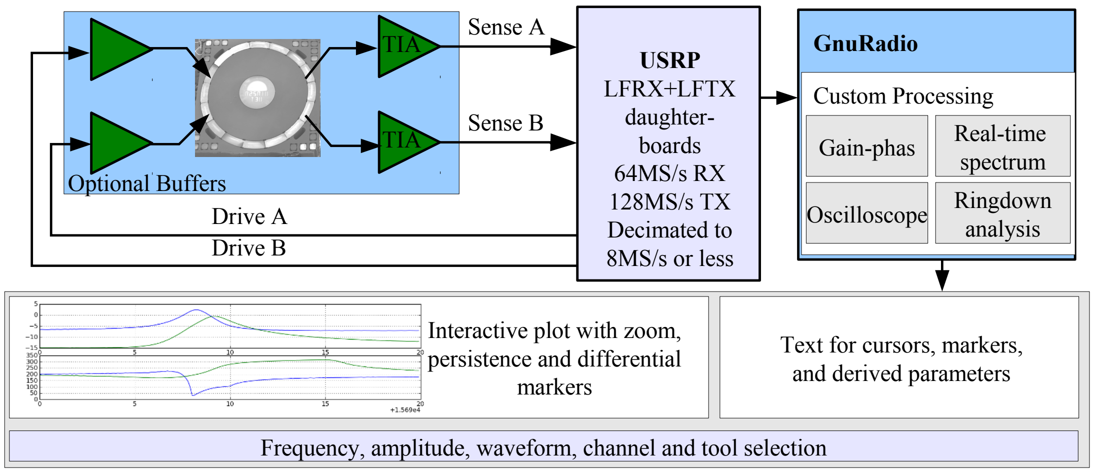

Sense A
Sense B
TIA
TIA
Optional Buffers
Drive A
Drive B
USRP
LFRX+LFTX daughter-boards
64MS/s RX
128MS/s TX
Decimated to 8MS/s or less
GnuRadio
Custom Processing
Gain-phas
Real-time spectrum
Oscilloscope
Ringdown analysis
Interactive plot with zoom, persistence and differential markers
Text for cursors, markers, and derived parameters
Frequency, amplitude, waveform, channel and tool selection

Figure 3.1: Diagram of the Universal Resonator Analysis Tool (URAT). URAT combines gain-phase analysis, real-time spectrum analysis, pulse generation, and ring down testing.

implemented software called the Universal Resonator Analysis Tool (URAT), Figure 3.1. URAT can operate simultaneously on two channels in swept-frequency gain-phase analysis, real-time spectral analysis with band-limited pulse generation and ring-down analysis modes. Dual channel operation is especially important for coupled resonator devices such as vibratory MEMS gyroscopes. The same hardware, without any recabling, also performs control functions for inertial sensing, in Chapters IV and V.

The sensitivity and performance of MEMS resonant sensors is directly linked to the quality factor and frequency of the resonator. In gyroscopes for instance, sensitivity can be increased by matching the resonant frequencies of the drive and rate-sense axes so that the sensitivity S is

$$S \propto A_g \frac{Q}{\pi f} = A_g \tau$$

where A_g is the mechanical gain, Q is the quality factor, f is the resonator frequency and τ is the damping time constant. Characterization and tuning of these resonators

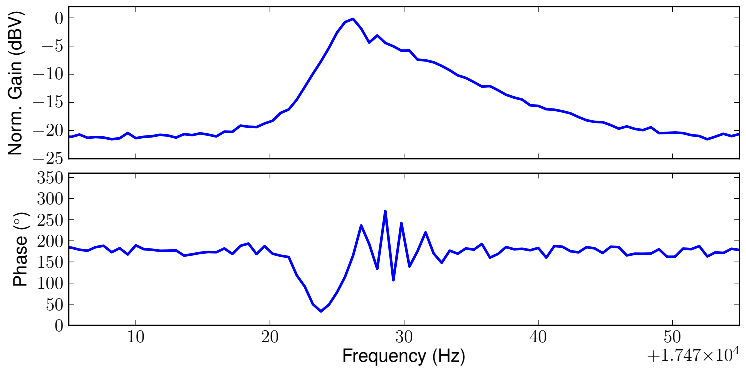


Figure 3.2: Gain-phase analysis result with frequency swept too quickly. There is interference between oscillations from tested frequencies which are still decaying and the current frequency, resulting in a distorted peak. This makes it difficult to characterize the peak or identify closely spaced peaks.

presents several special challenges. Because the bandwidth f/Q is small, long sampling times are required to provide sufficient resolution with real-time spectrum techniques. For swept-frequency characterization, drive times significantly longer than the damping time constant $Q/\pi f$ are necessary to avoid distorting the peaks as shown in Figure 3.2.

3.1 Universal Resonator Analysis Tool

The Universal Resonator Analysis Tool (URAT), Figure 3.1, was developed to address several challenges of testing high-Q, low-frequency gyroscopes with a limited budget. The most important limitation of available resonator analysis hardware is that most are single channel, while the gyroscopes are effectively two coupled resonators that need to be characterized simultaneously. For instance, to characterize the frequency mismatch of the gyroscope, the normal procedure is to drive the gyroscope under test with an off-axis electrode and try to sense both modes on a single sense electrode. Another method is to rely on the cross-axis coupling of the gyroscope to produce both modes on one sense electrode. These methods require different drive and sense configurations for resonance characterization and rate measurement or limit a tuning methods ability to cancel cross-axis coupling before attempting to match the mode frequencies. URAT with a USRP can drive and sense two channels simultaneously which allows simultaneous measurement of multiple axes and reduces measurement time.

A second limitation of commercial test equipment is that it tends to be single purpose. One set of equipment is used for slow swept-frequency measurements, another for real-time spectrum analysis, and a another setup for ring-down analysis. URAT performs all of these functions with the same hardware and software interface, and can be switched between functions using a menu. Rapid identification of peaks can be done using the impulse-RTS tool, and then the software switched to ring-down mode to check the damping time-constant of the peaks.

Lastly, resonator characterization equipment available on the market is too expensive to have multiples of each tool. The hardware used for the URAT is available commercially for under \$1000 and the software can be installed to a USB drive to turn any computer into a powerful resonator characterization tool. A brief summary of some commercial tools available is given in Table 3.1. The performance of URAT at low frequencies, below 1MHz, is similar to the commercial tools but more convenient and at much lower cost. Very few gyroscopes are designed to work at frequencies over 1MHz, and most operate from 1kHz to 100kHz. With some redesign, such as moving the mixing of signals to the FPGA, URAT could be expanded up to about 32MHz. Operation at even higher frequencies would require a change in the SDR hardware or a more sophisticated analog front-end.

URAT implements four different tools, swept-frequency gain-phase analysis (Section 3.1.1), real-time spectral analysis (Section 3.1.2), ringdown analysis Section (3.1.3), and an oscilloscope mode for checking signal quality. To highlight the capabilities of the URAT tool, an automatic mode tuning algorithm has been implemented and is described in Section 3.2. The URAT tool consists of a common user

Table 3.1: Comparison of Resonator Analysis Tools

	URAT (this work)	E5061BEP-NZA [42]	RSA5103A [43]
Function	Gain-Phase, RTS, ring-down	Gain-Phase, Imped., S-Param	RTS, Transient
Frequencies	≈DC-1MHz	5Hz-30MHz	1Hz-3GHz
GP Ports	2+	1	1
Sweep	Up, down, impulse	Up	External
Cost	< \$1000	>\$35,000	>\$30,000

interface for all of the applications, custom back end blocks that implement the various functions, and some analysis code to calculate the quality factor and mismatch characteristics of coupled resonators.

3.1.1 Swept-frequency Gain-Phase Analysis

Swept-frequency gain-phase analysis is basically a lock-in amplifier which sweeps the source frequency and records both the amplitude and relative phase of the received signal. It is implemented in URAT as shown in Figure 3.3. A software block generates an IQ (cosine and sine) reference signal and a test signal of a user selected waveform. The test signal frequency can be an arbitrary multiple of the reference signal frequency for testing non-linear devices. Waveforms which have been implemented include sine (both outputs in-phase), IQ (90° out of phase), differential (180° out of phase), triangle, and saw-tooth along with other custom waveforms for testing various driving schemes. Currently, the two output channels always have the same amplitude but either can be disabled from the URAT interface. The waveform is constructed in software at an intermediate sampling frequency (typically 500 kS/s, and up to 4 MS/s) and then is transferred to the USRP via USB where it is interpolated to 64 MS/s and transmitted to the CODEC where an additional interpolation brings the final signal rate to 128 MS/s. For capacitively driven resonators, the raw output of the USRP with an LFTX is generally sufficient and it can be around $3V_{pp}$. Additional

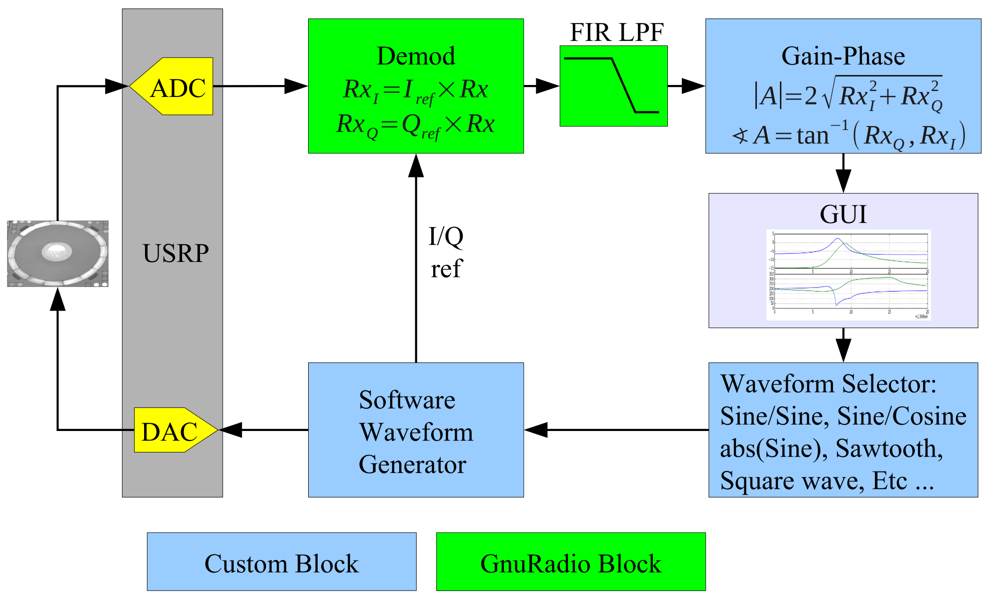


Figure 3.3: Diagram of gain-phase analyzer. Gain-phase analysis is appropriate for resonators with medium or short damping times. The drive frequency is swept over a defined range and the sense signals are demodulated at the drive frequency to determine the gain and relative phase.

external drivers can be added to boost the signal or generate multiple differential drive signals.

On the receive side, the signal is sampled by dual 64 MS/s 12 bit ADCs. For capacitively sensed resonators, it is generally necessary to include some external signal conditioning circuit, such as a transimpedance amplifier as in Figure 2.9. The maximum input signal is $2V_{pp}$ with programmable gain from 0 to 20 dB. With 20 dB gain the maximum input is then 200 mV_{pp}. In the USRP, the signal is decimated by a programmable amount, typically to 500 kS/s for our applications, and transferred by USB to the host computer. The signal interpolation, decimation and filtering in the USRP are all standard functions of the USRP.

In the URAT software, the signal from each channel is multiplied by the sine and cosine references and then decimated and filtered. The resulting baseband I,Q signals are then used to calculate the relative gain and phase of the two channels. The gain

Table 3.2: URAT input and output amplitude mismatch.

Input mV	Rx Gain dBV	Mismatch dBV	Output mV	Tx Gain dBV	Mismatch dBV
50	0	0.03	850	-6	.18
50	6	0.10	600	-9	.33
50	12	0.13	425	-12	.37
50	20	0.12			
100	20	0.10			

and phase are calculated according to

$$\begin{aligned} |H(f)| &= 2\sqrt{I \times I + Q \times Q}/A \qquad \text{Gain} \\ \measuredangle H(f) &= \arctan(Q/I) \qquad \text{Phase} \end{aligned} \tag{3.1}$$

where A is the output amplitude and arctan is implemented as an extended arctangent to generate signals between 0° and 360°. The gain and phase are averaged over a period set in the user interface which is also the amount of time the drive signal is applied. For devices with large damping times, this may be several seconds in order to clearly resolve the peak. The current system is very good at determining the relative gain and phase over frequency and between the two sense channels. The amplitude matching is summarized in Table 3.2. The best input mismatch is with 0 dBV RX gain, and is below 0.15 dBV for all measured combinations. This mismatch was measured by feeding the same signal to the front-end and reflects the mismatch in the daughter-board and ADC. The output mismatch is slightly larger, but still below 0.4 dBV.

The main limitations of the current system are the lack of a calibration step or an external reference signal. The phase of $H(f)$ is a function of the delay in the system which, because of the implementation of the USB buffers in software and USRP, can change occasionally. A calibration step would allow the delay to be measured and the phase measurement corrected, but since the delay can change, calibrations would have to be performed frequently. With an external reference signal, feeding the output

Table 3.3: URAT gain-phase analysis compared to HP4194A

		HP4194A			URAT		
Mode	Drive, Sense	f_0 kHz	Q	Gain dBV	f_0 kHz	Q	Gain dBV
1	1,1	20.6605	45.6	0	20.6603	45.1	0
1	1,2	20.6603	55.2	-3.5	20.6603	53.1	-3.3
1	2,1	20.6604	58.9	-5.1	20.6603	52.5	-4.9
1	2,2	20.6603	56.6	-7.9	20.6603	58.9	-8.1
2	1,1	20.6978	28.9	-10.8	20.6972	18.7	-11.2
2	1,2	20.6978	49.4	-5.8	20.6973	45.0	-6.4
2	2,1	20.6976	46.7	-6.6	20.6972	41.1	-6.5
2	2,2	20.6977	48.7	0.4	20.6973	47.0	0.5

back to a reference input, the phase and amplitude could be determined absolutely and reliably. Since we are currently using only two of the four ADCs and DACs of the USRP1 this is a possibility if future applications require it.

To evaluate the accuracy of the URAT gain-phase anaylsis tool, we compared it with an HP4194a operating in gain-phase mode, the results of which are in Table 3.3. The measurement was on a $\approx$20 kHz gyro with $\approx$40 Hz frequency split between the mode peaks. The device was tested with all four combinations of drive and rate-sense axes for each mode. These 8 measurements took approximately 40 minutes with the HP4194a since the cabling needed to be changed for each measurement, and less than 4 minutes with the URAT since only 2 measurements were required for each peak and no cables needed to be changed. The shorter test time with the URAT may explain the reduced variation in the resonance frequencies.

3.1.2 Impulse Generator and Spectrum Analysis

The second URAT tool is a real-time spectrum analyzer with a band-limited impulse generator (impulseRTS) illustrated in Figure 3.4. The impulse generator generates the desired test signal in the frequency domain with power only in the frequencies of interest, and then generates the time domain signal by applying an

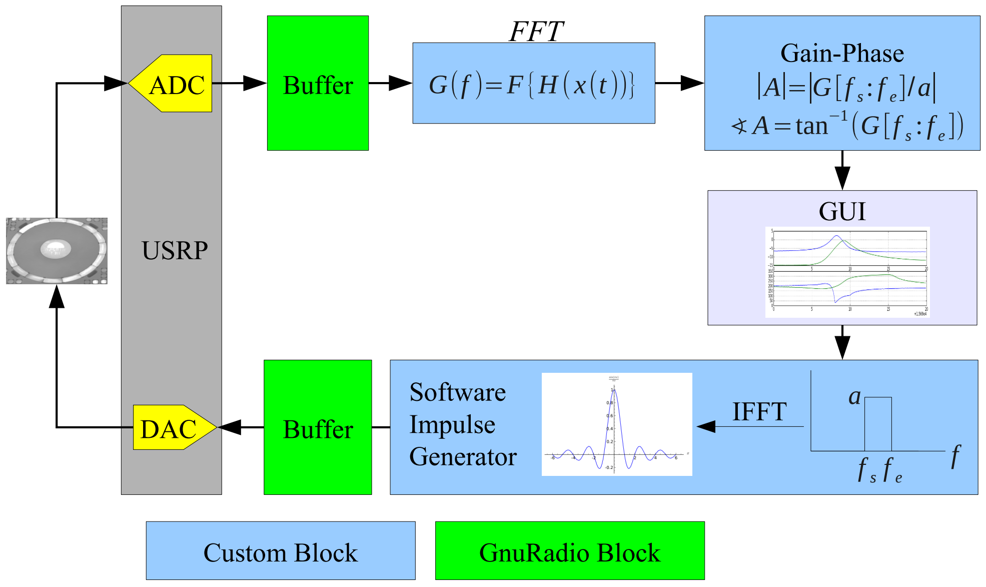


Figure 3.4: Diagram of real-time spectrum analyzer with impulse generator. Real-time spectrum analysis is most useful for quickly analyzing a moderate bandwidth. Band-limited pulses are generated to excite the resonator in the region of interest.

inverse Fourier transform. The FFT and IFFT are both from the SciPy [37] library. The time domain signal is transferred to the USRP and, since the drive and sense time windows will not align exactly, repeats until the required sampling time for the specified frequency resolution has expired.

The response is sampled as in Section 3.1.1 and then processed on the host computer where the FFT is taken of the whole signal and then the gain and phase are calculated for only the frequencies which were non-zero in the source spectrum. The gain and phase are calculated according to

$$\begin{aligned} G &= \frac{\text{fft}(Rx \times \text{window})}{N} \\ |H(f)| &= \frac{\sqrt{\Re\left[G[f_s:f_e] \times G[f_s:f_e]^*\right]}}{G_{ref}} && \text{Gain} \\ \measuredangle H(f) &= \Re\left[\text{angle}(G[f_s:f_e])\right] && \text{Phase} \end{aligned} \tag{3.2}$$

where Rx is the sample signal, fft and angle are functions for the FFT and angle of a complex number from the SciPy library, window is a windowing function which defaults to a hanning window (also from the SciPy library), f_s and f_e are the start and end frequencies, and G_{ref} is the source frequency spectrum magnitude.

The result of testing with the impulseRTS tool for a gyroscope before and after mode matching is shown in Figure 3.8, and the impulseRTS tool was used for the automatic tuning as described in Section 3.2. The impulseRTS tool is useful when looking at a relatively wide bandwidth with moderate frequency resolution, because it is much faster than the swept-frequency tools. Also, because it tests all frequencies simultaneously, peaks with long damping times do not obscure other peaks as is the case with the swept-frequency tools and so closely spaced high-Q peaks can be clearly identified.

The frequency resolution in an FFT is limited by the sampling time, which in the current implementation is limited by the system memory. Since we are usually interested in only a small part of the bandwidth, other Fourier transform techniques, such as zoom-FFT or sparse-FFT, could reduce the processing time and improve the frequency resolution, which is currently limited to about 50 mHz. The impulse generator is good for devices which have long damping times (0.1 s or more), which is typical of high-Q, low-frequency MEMS. These devices will have good signal to noise ratio. For devices with lower damping times, a chirp signal would be better since the maximum energy at each frequency is higher and the signal will decay within the measurement window even if the resonant frequency is close to the end of the chirp. Both chirp and simple averaging have been implemented as options and are especially useful when working with a Laser Doppler Vibrometry (LDV) which can have poor SNR. Complex averaging requires a reference signal which is not currently implemented.

Figure 3.5 shows the frequency response of a resonator with an n=3 elliptical mode around 325.6 kHz with a 200 Hz mode-split. Eleven averages were used in this

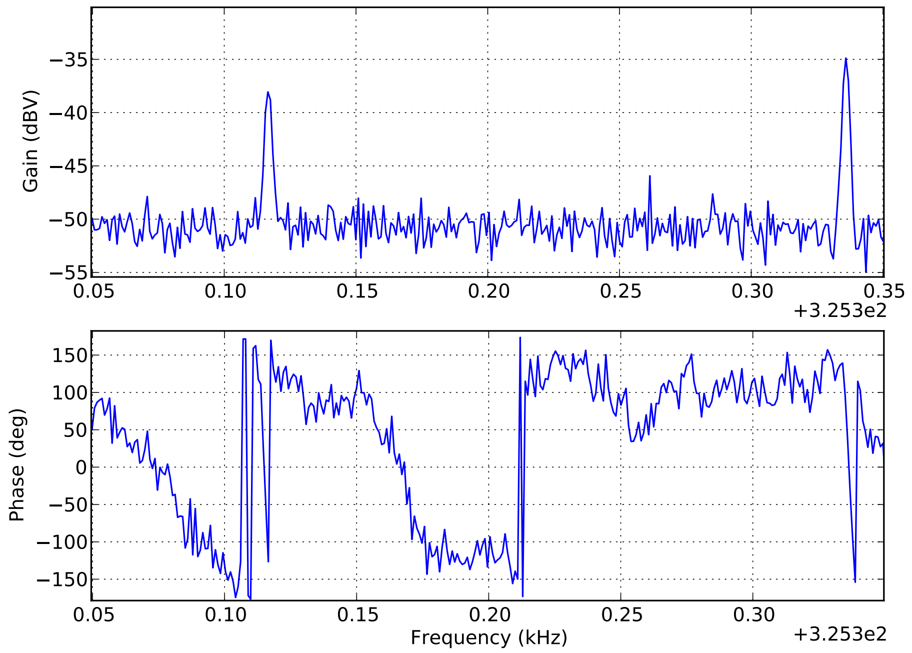


Figure 3.5: Plot of impulse testing of a high frequency resonator. The use of impulse testing significantly reduces the effect of feed-through which is especially significant at higher frequencies.

measurement with 1Hz resolution resulting in the entire 700Hz measurement taking about 12 seconds.

3.1.3 Ringdown Test Tool

Ring-down time is defined as the time required for the oscillation of a resonator to decay from A_0 to A_0/e and is equivalent to the damping time constant τ. The ring-down analysis tool is for measuring the ring-down time of resonators where the ring-down time ranges from a fraction of a second to hundreds of seconds. The drive signal generation and signal sampling are identical to that described in Section 3.1.1, however the received signal is feed to a software PLL which generates I,Q reference signals which are used to demodulate the received signal and calculate the gain. After a predetermined amount of time, which should be several times longer than the damping time, the drive signal is disabled and the resonator begins to ring down.

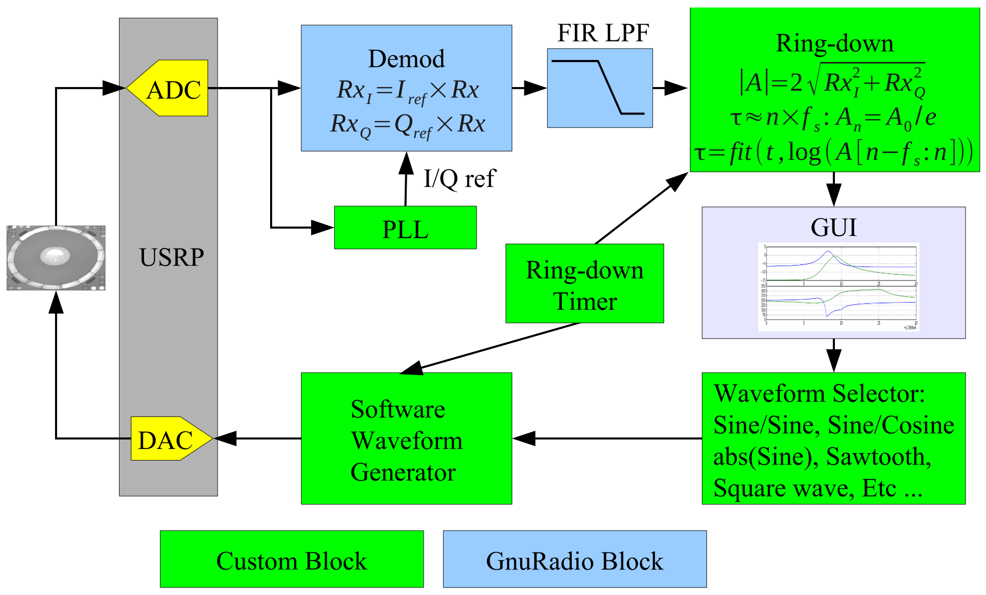

ADC
USRP
DAC
Demod
$Rx_I = I_{ref} \times Rx$
$Rx_Q = Q_{ref} \times Rx$
FIR LPF
Ring-down
$|A| = 2\sqrt{Rx_I^2 + Rx_Q^2}$
$\tau \approx n \times f_s : A_n = A_0 / e$
$\tau = fit(t, \log(A[n - f_s : n]))$
I/Q ref
PLL
Ring-down Timer
GUI
Software Waveform Generator
Waveform Selector: Sine/Sine, Sine/Cosine abs(Sine), Sawtooth, Square wave, Etc ...
Custom Block
GnuRadio Block

Figure 3.6: Diagram of ring-down analyzer. Ring-down analysis is appropriate for devices with long damping times. A drive signal is applied for a set amount of time and then removed. A PLL locks on to the sense signal and ensures that it is demodulated in-phase even if the frequency shifts during ring-down.

Since the drive signal is not at exactly the resonance frequency, the frequency of the resonator may drift during ring down which is why the PLL is used instead of reference signals from the waveform generator.

The damping time constant is initially approximated by the time required for the signal amplitude to decay to the initial signal amplitude over e. Once the signal has decayed, the software waits for a configurable additional delay and then fits an exponential decay function to the middle of the ring-down curve, as in Figure 3.7.

The frequency of the ring down is swept because the quality of the fit depends on the SNR of the signal which will vary considerably with even small offsets from the resonance frequency for high-Q resonators. Also, for nearly-matched coupled resonators, the peaks may be very close but not quite identical. Closed loop self resonance circuits or fixed frequency tests would have difficulty distinguishing the peaks.

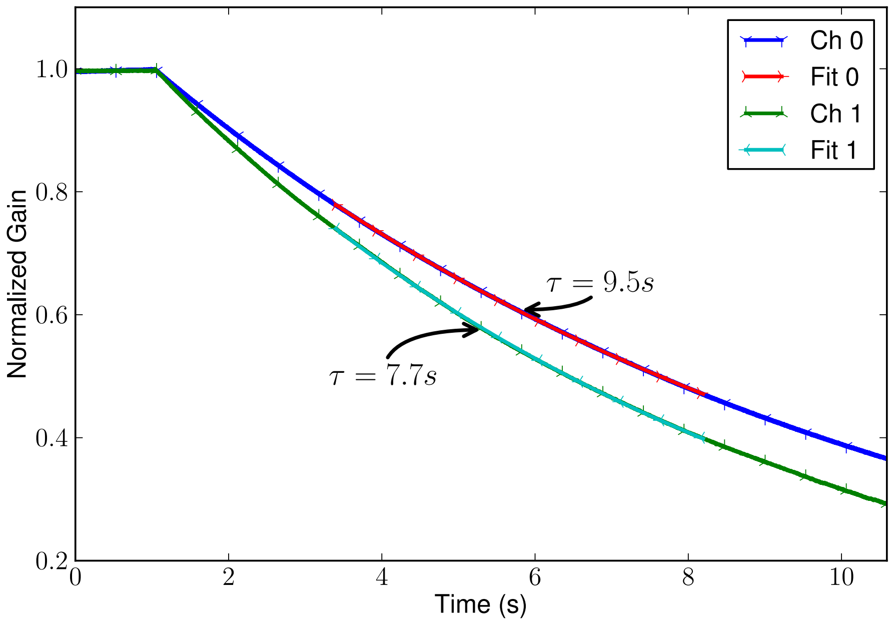


Figure 3.7: Plot of ring-down of a mode-matched CING gyro. The exponential decay fit is overlayed on top of the decay curves. Each curve is the normalized envelope of the received signal at the test frequency, which is recovered by demodulating the signal.

3.2 Automatic Mode Matching

Mode matching (matching the rate-sense and drive axes resonance frequencies) improves the sensitivity of rate gyroscopes and allows operation in rate-integrating mode. Several real-time methods of mode matching [21, 26] have been presented which are suitable when the initial mismatch and cross-coupling are small. In URAT we have implemented automatic mode tuning meant to provide a 'coarse' tuning that will provide the starting point for dynamic mode matching techniques for use during gyroscope operation. The automatic tuning of the CING is based on the analysis in [25], where they give an equation for the spring softening due to arc shaped electrodes around a ring. Other tuning electrode schemes can be implemented by replacing the stiffness matrix model. The equations for the spring-softening matrix as used in this work are given in Equation 3.3, where ϵ_0 is the vacuum permittivity, a is the electrode radius, d is the electrode height electrode area and h_0 is the electrode gap, ψ is the electrode position in radians, and α is the electrode width in radians. The tuning electrodes for the CING are under the CING rather than around it, however this is expected to only effect C_k, which is determined experimentally. The reliability and speed of the algorithm depends on the ability of the software to accurately determine the mismatch parameters.

$$C_k = \frac{\epsilon_0 a d}{2h_0^3} \qquad [K] = C_k V_i^2 \times \begin{bmatrix} 2\alpha + \frac{1}{n}\cos 2n\psi \sin 2n\alpha & \frac{1}{n}\sin 2n\psi \\ \frac{1}{n}\sin 2n\psi & 2\alpha - \frac{1}{n}\cos 2n\psi \sin 2n\alpha \end{bmatrix} \tag{3.3}$$

The gyroscope is scanned using one of the analysis tools, usually the impulseRTS tool (Section 3.1.2) but the gain-phase analyzer also works, and the mismatch parameters are modeled as spring constants which are calculated as

$$\begin{aligned}
[K] &= \begin{bmatrix} k + \Delta k/2 & k_{xy} \\ k_{yx} & k - \Delta k/2 \end{bmatrix} \\
k &= (2\pi \frac{f_1 + f_0}{2})^2 \\
\Delta k &= 4\pi^2 (f_1 - f_0) \frac{f_1 + f_0}{2} \\
k_{xy} &= s_k 2 \Delta k \max[H_0(f_1)/H_1(f_1), H_1(f_0)/H_0(f_0)] \\
s_k &= \begin{cases} -1, & \text{if } (\phi_0(f_0) - \phi_1(f_0)) > \pi/4 \\ 1, & \text{otherwise} \end{cases}
\end{aligned} \tag{3.4}$$

where f_0 and f_1 are the frequencies of the the peaks of each mode and $H_n(f)$ is the gain of channel n at frequency f. In this model, everything is actually a frequency mismatch and whether the source is stiffness or mass anisotropy is not considered. In order to get matching better than the minimum resolution of the impulse-RTS tool, the peak frequencies are calculated using an amplitude weighted average of the FFT bins around the bin with the peak rather than just the index of the bin with maximum signal. It is also important to implement an algorithm that reliably identifies the peaks when they are closely spaced.

An initial guess of C_k is used to calculate a cross tuning voltage to cancel the k_{xy} terms and then the gyro is scanned again. The actual C_k is then calculated from the change in the measured mismatch and the change in the electrical spring model as

$$\begin{aligned}
[\Delta K_{meas}] &= \begin{bmatrix} \Delta k/2 & k_{xy} \\ k_{xy} & -\Delta k/2 \end{bmatrix} \\
[\Delta K_{el}] &= K_{el,prev} - K_{el,new} \\
A &= [\Delta K_{meas}][\Delta K_{el}^{-1}] \\
C_k &= \mathrm{tr}(A)/2
\end{aligned} \tag{3.5}$$

where $K_{el,prev}$ is the electrical spring model with the previous tuning voltages and $K_{el,new}$ with the new tuning voltages, and tr is the trace of the matrix. If the tuning is in a linear regime, A will be the identify matrix times C_k. Error in the measurement and non-linearity however create off-axis terms. The relative size of these terms is a measure of the quality of the estimate of C_k. To avoid having the tuning fail due to erroneous values of C_k, the update of C_k can be performed as

$$\begin{aligned} g &= 1 - \mathrm{tr}(A^T)/\mathrm{tr}(A) \\ C_k &= \begin{cases} g^2\mathrm{tr}(A)/2 + (1-g)^2 C_{k-}, & \text{if } g > 0 \\ C_{k-}, & \text{otherwise} \end{cases} \end{aligned} \tag{3.6}$$

where g is the 'goodness' of the prediction and C_{k-} is the previous value of C_k.

If k_{xy}/k is larger than a configurable threshold, a new cross tuning voltage is chosen based on the new estimate of C_k. This processes is iterated until the threshold is passed and then the algorithm tunes $\Delta k/k$ until the frequency mismatch threshold is passed. The process is then repeated again for k_{xy} and Δk with lower thresholds. A typical tuning run for a 3kHz gyro with a damping time of several seconds will take a few minutes, 6-10 steps, and tune the frequency mismatch to a few tens of mHz. The spectrum before and after for a CING gyro are shown in Figure 3.8. With this automatic tuning procedure, the drift in the tuning voltages over time is easily observed as in Figure 3.9.

The algorithm still has some weaknesses. The measured Δk and k_{xy} versus tuning voltage are plotted in Figure 3.10 and C_k is related to the slope of the curves. If the gyro is over-tuned, meaning too large of tuning biases are applied, the mechanical and electrical spring constants are no longer linear and the calculations for C_k can produce erroneous values. Also, damping mismatch will influence the calculation for k_{xy} and for gyros with large damping mismatch it can be difficult for the auto-tuning to work reliably. Ideally C_k would be constant, however there is variation, especially

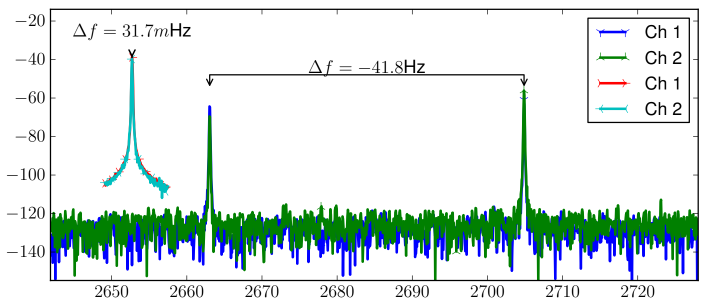


Figure 3.8: Plot of the initial and final spectrum of an automatically mode-matched CING gyroscope. The mode matching took 6 steps and generally takes about 3 minutes for the lower frequency gyroscopes.

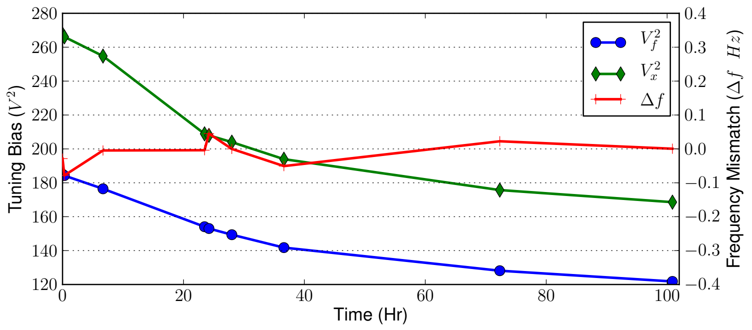


Figure 3.9: Drift in required tuning voltage for a CING over time. The CING gyro has a large drift in required tuning voltages for matched conditions which eventually settle. This is easily tracked using the automatic mode matching algorithm. In the above plot, the gyroscope was retuned 9 times over 4 days. If the tuning voltages are removed, the process will reverse and over time the required tuning voltages will move back up the curve.

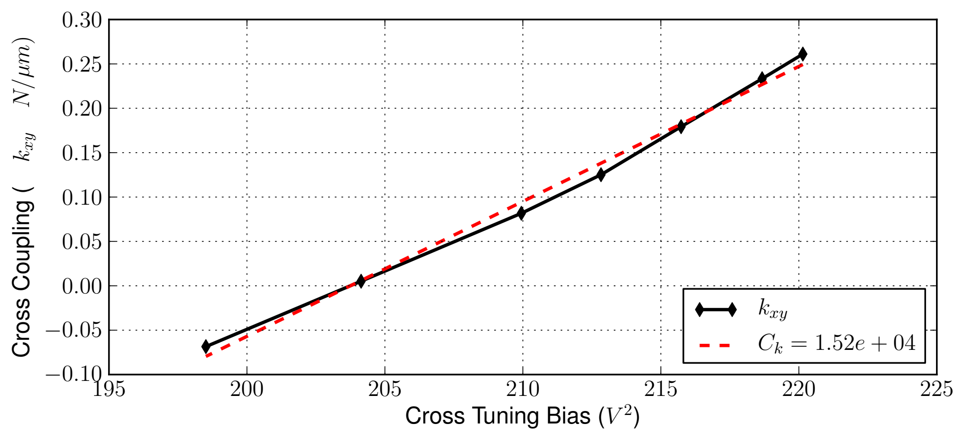


Figure 3.10: Cross-coupling versus tuning voltage. Initially the cross coupling is negative. As the tuning bias is increased, the cross coupling becomes nonlinear as it is overtuned. The dashed line is a linear fit of the data showing the effective C_k is 15,200.

for the cross-tuning terms as shown in Figure 3.11.

3.3 URAT and Laser Doppler Vibrometry

The URAT tool can also be used for non-electrical testing. LDV is an optical method to detect the velocity of a structure from the shift in the frequency of reflected light due to the Doppler effect. The test can be set up as illustrated in Figure 3.12 with the USRP directly driving a small piezo actuator with a small resonator attached to it. For the actuator used here, the drive signals were only a few millivolts. Larger signals caused the oscillations to be unstable or would even cause the resonator to crack. Using URAT for this test provides access to tools for characterizing high-Q devices which aren't available in most LDV software packages, such as ring-down testing. The result of ring-down testing a resonator with a 2 s ringdown time is shown in Figure 3.13.

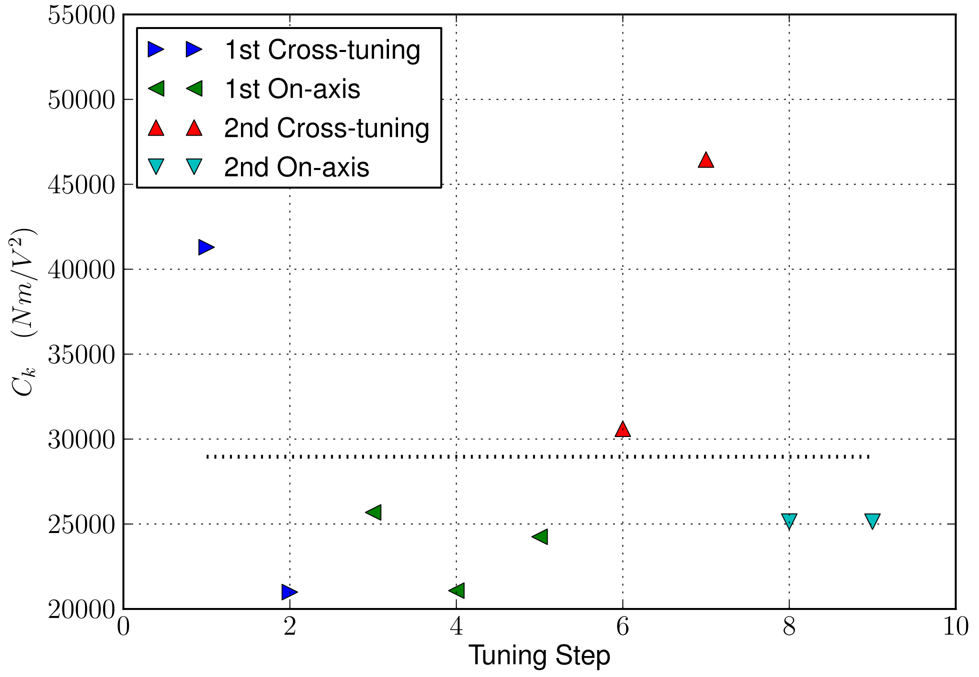


Figure 3.11: Tuning electrode scaling constant stability. The scaling constant is more stable for on-axis tuning than off-axis tuning. The damping mismatch will affect the accuracy with which the off-axis terms are measured which is probably introducing error in the calculation of C_k.

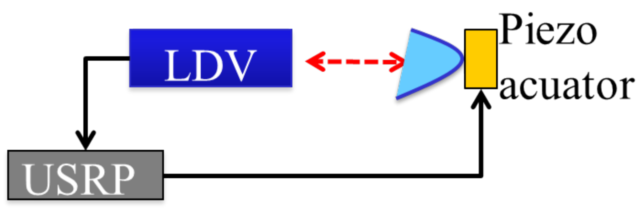


Figure 3.12: Setup for using URAT with a laser doppler vibrometer. Using URAT with a laser doppler vibrometer (LDV) extends the range of devices that can be characterized and adds special features like ring-down testing which are not available in most LDV software.

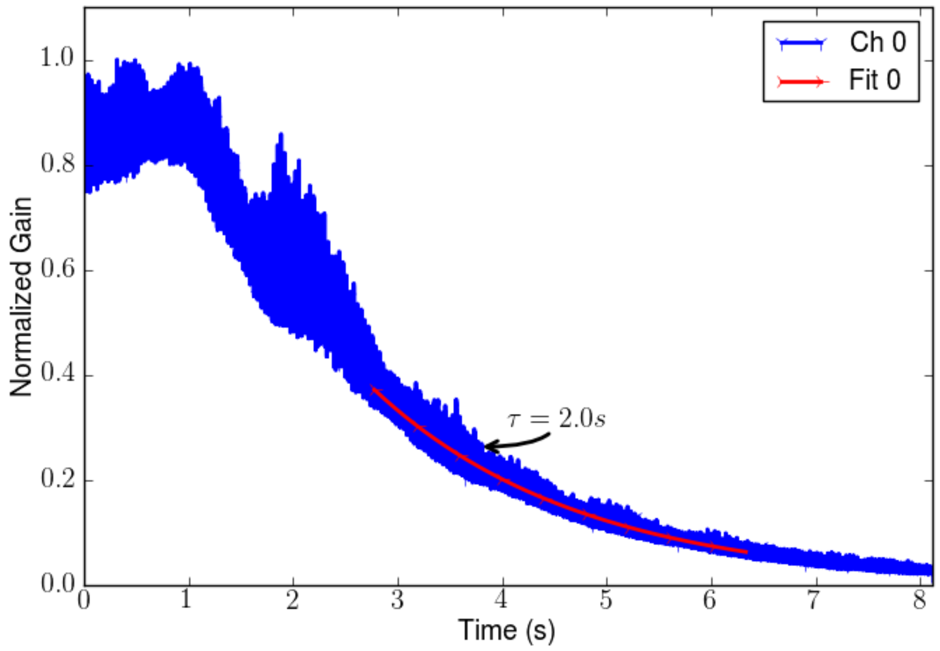


Figure 3.13: Ring-down measured using a laser doppler vibrometer. The signal-to-noise ratio for this LDV test was poor due to the structure being transparent, but it was still possible to get a reliable result using the ring-down analysis tool.

3.4 Summary

The contributions of the work presented in this chapter are a characterization system consisting of SDR hardware and software tools, and automatic mode-matching of MEMS gyroscopes. The URAT software provides gain-phase, impulse and real-time spectrum, and ring-down tools with performance similar to specialized commercial tools. These tools are especially well suited to analysis of devices with long damping times, over 0.1s, or high-Q and low-frequency. Since these tools are in the same software, it is easy to switch between them depending on the characterization task, which can greatly decrease test time or errors due to switching equipment. In addition to standard electrical testing, URAT has also been used with a LDV for testing devices without electrodes.

CHAPTER IV

Rate Gyroscopes

Rate gyroscopes are a class of MEMS gyroscopes where the output is a signal proportional to the rate of rotation. Most MEMS gyroscopes are rate gyroscopes. The control system for a rate gyroscope needs to ensure that the amplitude on one axis, the drive axis, is constant. The second axis, or rate-sense axis, develops motion due to the Coriolis force, Equation 2.1, which is used to measure the rate of rotation.

4.1 Mechanical Noise

Noise in MEMS gyroscopes can be introduced by the electronics as described in Section 2.2.1, but there are also mechanical sources of noise which may be more significant. Mechanical noise is usually presented as Brownian motion [45] and is analyzed in extensive detail in [46] for both open-loop and closed-loop operation. The ARW for an open-loop gyroscope is

$$\Omega_{rw} \approx \sqrt{\frac{k_B T \omega_y}{A^2 m \omega_x^2 Q_y} \frac{1}{1+\omega_d^2 \tau_y^2}} \times 3437.7^\circ/\sqrt{Hr} \tag{4.1}$$

where k_B is the Boltzmann constant, T is the temperature, ω_y is the sense axis resonant frequency, ω_x is the drive axis resonant frequency, ω_d is the frequency mismatch $(\omega_x - \omega_y)/2$, A is the drive axis amplitude, m is the mass of the resonant mode, Q_y

is the sense axis quality factor, and τ_y is the sense axis damping time constant or $(2Q_y)/\omega_y$. The constant 3437.7 converts the noise from rad/$\sqrt{s}$ to °/$\sqrt{Hr}$.

The ARW in closed-loop operation is

$$\Omega_{rw} \approx \sqrt{\frac{k_B T \omega_y}{A^2 M \omega_x^2 Q_y}\left(1+\omega_d^2\tau_y^2\right)} \times 3437.7°/\sqrt{Hr} \tag{4.2}$$

which is identical to the equation for the open-loop except for the factor of $(1+\omega_d^2\tau^2)$ which is zero if the signals are matched and not significant if the frequency mismatch is much less than the damping time constant. Although the total rate equivalent noise (the RMS noise) is larger in the closed-loop mode due to the increased bandwidth, the integration of the rate to create an angle cancels this effect.

A resonator also has phase noise. One approximation of phase noise is given by [48]. The demodulated phase noise contributes to 1/f noise in the rate signal which determines the bias stability of the gyroscope. Phase noise is reduced with a high quality factor and stable bias voltages.

4.1.1 Feed Through

Drive signals and compensation signals are often much larger than the sense signals in MEMS gyroscopes. This feed-through is not a noise source, but can disrupt the phase locking and compensation of the control system. Feed-through is dominated by parasitic capacitance between the drive and sense nodes and so is phase shifted by 90°. The gyroscope is operated at resonance so that the phase shift is 0°. Drive feed-through on the drive axis therefore appears as quadrature error and phase error at the rate and drive sense channels. This error due to feed-through causes the controls to over or under compensate and therefore results in a non-zero steady state error.

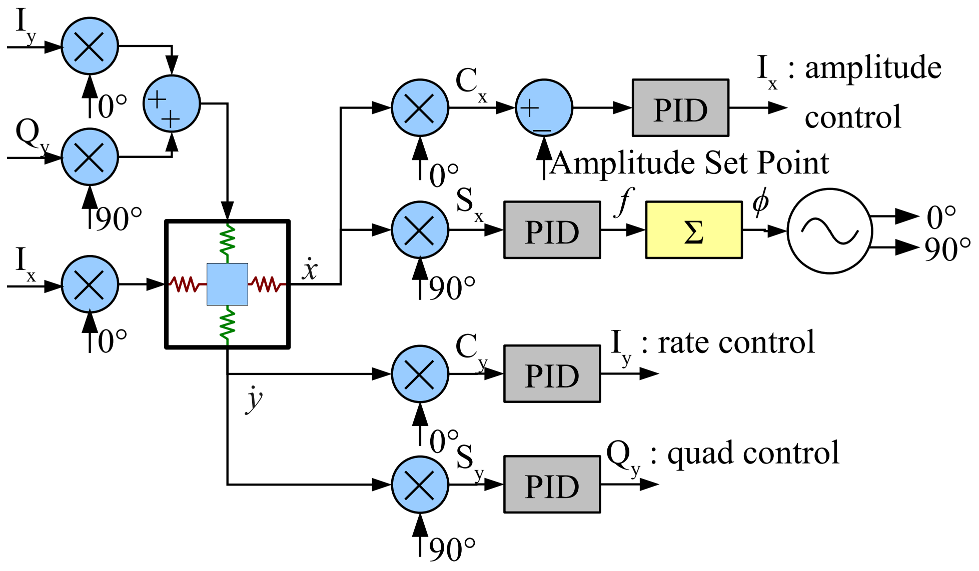


Figure 4.1: Diagram of rate gyroscope control with rate and quadrature feedback. The rate gyroscope control implemented in this work consists of the blocks shown here, similar to what is proposed in [17]. It is assumed that sense signals are proportional to the velocities, but using rate signals instead only requires changing the sign of the gains in the PIDs. There are paths for amplitude control, phase tracking, rate control and quadrature control.

4.2 Rate Control

4.2.1 Control Equations

The basic control loops for a MEMS rate gyroscope are amplitude control, phase control (a PLL), rate or force-feedback control, and quadrature control as illustrated in Figure 4.1. Strictly speaking, none of these controls are necessary, but by using all of them it is possible to improve the gyroscope performance. The control is based on the demodulated sense signals. The details of the demodulation depend on whether the output of the analog interface is in phase with the oscillation displacement (charge amplifiers and voltage mode amplifiers), or the oscillation velocity (transimpedance amplifier and current sensors).

Amplitude control is important for rate gyroscopes because the Coriolis acceleration is proportional to the amplitude of the drive axis. If the gyroscope is driven with

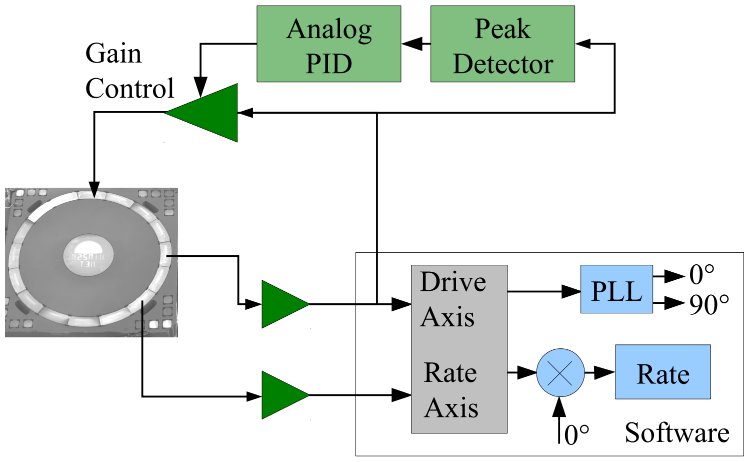


Figure 4.2: Diagram of a simple rate gyroscope analog control. The readout can be analog or digital as drawn here. This control has been implemented using analog circuits for the peak detector, variable gain amplifier and PID controller and using the USRP to digitize the signal and extract the rate information.

a fixed amplitude signal, the drive axis amplitude will settle to a value where either the drive signal is balanced by the damping or the amplitude non-linearity reaches a point where the drive is no longer effective. With an open-loop drive however, the drive amplitude will not be stable, especially under rotation. As the signal grows on the sense axis, the Coriolis force from the sense to drive axis will act as a damping force on the drive axis causing the amplitude to change which will in turn affect the amplitude of the Coriolis force. To keep the amplitude constant, an amplitude control loop is used. This loop can be analog or digital. A simple analog control for rate gyroscopes is illustrated in Figure 4.2.

For the amplitude of the oscillation on the drive axis to be constant, the drive force must equal the damping force and if the gyroscope is driven close to the resonance frequency the drive signal is then

$$V_{drive} \propto \frac{2}{\tau}\dot{x} = \frac{\omega^2}{Q}x \tag{4.3}$$

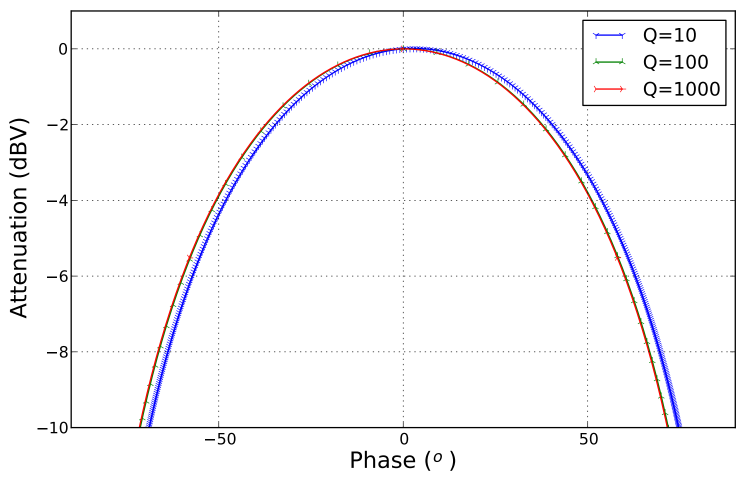


Figure 4.3: Resonator attenuation versus phase shift in drive loop. Driving a resonator away from the resonance frequency will cause an attenuation of the resonator gain that is nearly independent of the Q.

To efficiently sustain an oscillation on a resonator, it is necessary to drive it near the resonant frequency. One way to do this is with a closed-loop where the drive sense signal is amplified and fed back to the drive axis as in Figure 4.2. This loop will ensure that the gyroscope operates close to its resonance frequency where the phase shift is zero. The closed-loop can be either analog or digital, but it is important to minimize the phase shift to keep the resonator as close to the resonance frequency as possible, which is easier to do with an analog loop. The relationship between phase shift and the reduction of the resonator gain is plotted in Figure 4.3. This plot is made by calculating the gain and phase response of a resonator against frequency and then plotting the gain versus phase. Note that the reduction in the gain is independent of the quality factor when the quality factor is large. A small phase shift in the sustaining loop means the drive signal can be smaller which reduces non-linearity and feed-through.

A second method of creating a sustaining oscillator loop is with a PLL. In Figure 4.1, the PLL is formed by a demodulator which produces s_x which is proportional to the phase error, a PID which transforms the phase error into a frequency, an accumulator which integrates the instantaneous frequency to get the phase and a waveform generator which outputs in-phase (0°) and quadrature (90°) reference signals. Any extra phase in the system will cause the same attenuation as in the simple loop control above. Again, this method can be implemented with analog or digital electronics, but this approach is better than the sustaining amplifier loop for digital electronics because a digital PLL and digital filters can adjust the phase on the PLL to cancel the phase shift due to the filters. A second advantage of the PLL drive is that both the 0°and 90° reference signals are provided, which is important for the next two controls. One potential disadvantage of the PLL is that its lock-range may be much smaller than the start-up range with the simple loop amplifier. It is possible in a digital system to use a simple loop to start the oscillation and then switch to the PLL once the oscillation is large enough and the approximate resonance frequency has been determined.

The third control is a rate or force-feedback control. In Figure 4.2, there is no drive signal on the rate axis and so the amplitude on the rate axis grows or shrinks as the rotation rate changes. A signal can be applied to this axis to cancel any oscillation on it, so that the amplitude is always approximately zero. The amplitude of the control signal required to cancel the oscillation on the rate axis is then the rate signal. From Equation 2.2, the signal required to cancel the Coriolis force is

$$V_{rate} \propto 2A_g\Omega\dot{x}. \tag{4.4}$$

The advantages of rate control are improved bandwidth and linearity of the gyroscope response. The bandwidth is improved because the effective damping on the rate axis is increased by the control loop. The linearity is improved because the signal does not grow large enough to become non-linear. The downside is that the noise in the sensed rate is then fed back into the rate axis, which increases the noise by $\sqrt{2}$.

The fourth control is quadrature control. In a mode-matched gyroscope, the quadrature control is in phase with the resonator displacement and is used to cancel the signal at the rate-sense axis that is out-of-phase with the rate signal. Since the signal is in phase with the displacement, it modifies the resonant frequency of the axis and can be thought of as forcing the drive and rate axes to have the same resonant frequency. From Equation 2.2, it can be seen that to modify the resonant frequency of the rate axis to equal the drive axis (assuming $\theta_\omega = 0$), the quadrature signal must be

$$V_{quad} \propto \omega \Delta \omega x. \tag{4.5}$$

where $\omega\Delta\omega$ is defined in [19] as

$$\omega \Delta \omega = \frac{\omega_1^2 - \omega_2^2}{2} \tag{4.6}$$

but it is more convenient for analysis to write this is terms of the average resonant frequency $\bar{\omega}$ and the frequency difference $\delta\omega$ as

$$\omega \Delta \omega = \frac{(\bar{\omega} + \delta\omega)^2 - (\bar{\omega} - \delta\omega)^2}{2} = 2\bar{\omega}\delta\omega \tag{4.7}$$

where $\delta\omega = \omega_1 - \omega_2$.

The range of rates and frequency mismatch that the rate and quadrature control can compensate for may be understood by normalizing the voltages required to the drive voltage. The drive voltage effectiveness is amplified by the quality factor, where as the rate or quadrature signals are not proportional to the quality factor. The normalized rate signal is then

$$\frac{V_{rate}}{V_{drive}} = \frac{2A_g \Omega \dot{x}}{2/\tau \dot{x}} = A_g \tau \Omega \tag{4.8}$$

assuming that the gyroscope is symmetric so that the sense axis and drive axis capacitance is the same. The normalized quadrature signal is

$$\frac{V_{quad}}{V_{drive}} = \frac{2\omega\delta\omega x}{x\omega^2/Q} = \frac{2Q\delta\omega}{\omega} = \tau\delta\omega = \tau\pi(f_1 - f_2). \tag{4.9}$$

From these equations, the maximum rate and frequency mismatch for a gyroscope with a given drive amplitude, resonant frequency, damping and angular gain can be calculated from the voltage limit of the drive circuit. From the analysis in Section 2.2.2, it is clear that the third method where a bias voltage is used along with a drive voltage at the resonance frequency, gives the largest dynamic range and compensation capability. From Equations 4.9, the quadrature signal will be large even for relatively small frequency mismatch. For a 10 kHz ring gyroscope with $A_g = 0.3$, 1 Hz mismatch and a quality factor of 100,000, the quadrature compensation signal is 10 times larger than the drive signal. If the drive signal is $1/10^{\text{th}}$ of the full-scale range, then the maximum rate for that gyro is 10.5 rad/s or $600°/s$.

4.2.2 Mechanical Requirements

Not all gyroscopes are suitable for the controls presented here. These controls assume a gyroscope that is nearly mode-matched, so the natural mechanical modes must have nearly equal frequencies. Also, many gyroscopes only have electrodes to actuate the drive axis and sense the oscillation on the drive and sense axes. Closed loop control of the rate requires an electrode to actuate the sense axis as well. Quadrature control as presented here uses forces applied at the resonance frequency to cancel frequency mismatch, but the output of the quadrature loop could also be routed to a DAC controlling a DC voltage on a specially designed quadrature compensation electrode. Quadrature control is more effective however when combined with closed loop control of the rate oscillation. Gyroscopes which operate in the wineglass or n=2 mode, such as the CING, need an extra set of electrodes in order to have enough electrodes for fully differential tuning, driving, and sensing. If the gyroscope operates at the n=3 mode, such as [49], a single set of electrodes is sufficient since there are six antinodes for each mode which can support 2 tuning electrodes, 2 actuator electrodes and 2 sense electrodes).

The controls here are digital and use a digital PLL to construct the drive signal. The PLL is a finite frequency resolution. If f_0/Q for the gyroscope is close to the frequency resolution, the effective quality factor of the gyroscope will be reduced. Also, the quality factor of the gyroscope is a factor in the loop gain of the controls. Since the gains are applied using fixed point math, there is a minimum gain and if the quality factor is too high this minimum gain may be too high for stable operation. External attenuation or adjusting the fixed point scaling can address this problem.

4.2.3 Firmware

The same hardware, without any recabling, is used for both characterization and gyroscope control by changing the firmware and software. Control may either be implemented within the FPGA of the USRP, or in software with the FPGA implementing a PLL and modulating control signals and demodulating sense signals. Both of these approaches reduce critical delay in the self-oscillation loop, and the hybrid control allows advanced controls to be prototyped in software. The FPGA based control is discussed next and hybrid control is presented in Section 5.6.

4.2.4 FPGA Based Rate Control

The rate control firmware in Figure 4.4 implements a PLL and control loops for the amplitude, rate-sense axis force feedback, and quadrature compensation. To allow all of these to fit on the FPGA, one IIR filter and one PID are multiplexed between each parameter to be controlled. The gyroscope signals are demodulated using the CORDIC block distributed with GnuRadio and the output of the PLL

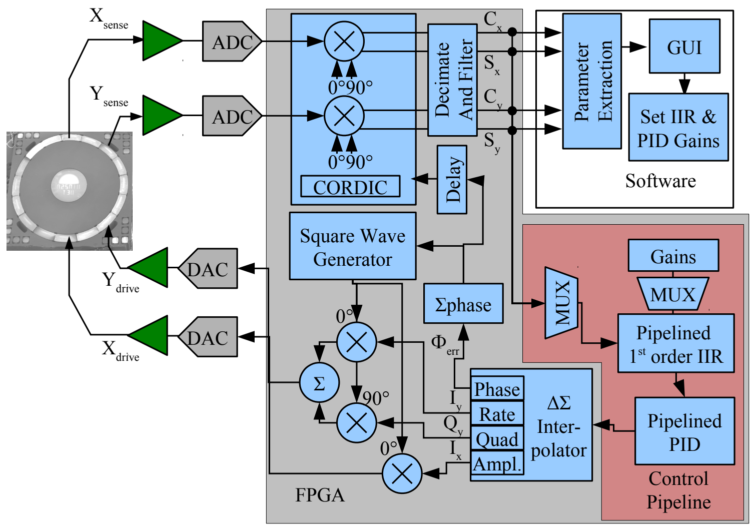


Figure 4.4: Diagram of rate gyroscope readout and control as implemented in the FPGA of the USRP1.

phase accumulator. CORDIC is an implementation of complex multiplication using adders, a full explanation of which can be found in [50]. The output of the two CORDIC blocks are decimated and filtered to construct the in-phase and out-of-phase signals for each axis, referred to as C_x, S_x, and C_y, S_y respectively. C_x represents the drive amplitude and is routed to the PID during the amplitude control step. S_x is proportional to the phase error of the PLL and is routed the PID during the phase control step, implementing a phase-locked loop. C_y is the rate signal when the gyroscope is mode matched and is routed to the PID during the rate (force-feedback) control step. S_y is the quadrature error signal, which is a function of the frequency mismatch between the drive and sense modes, and is routed to the PID during the quadrature compensation step. Each of C_x,S_x,C_y,S_y and the outputs of the PID during the four steps (I_x,ϕ_{err},I_y,Q_y) are routed to the USB transfer block and transmitted to the host computer. ϕ_{err}, the output of the phase control step, is also routed to a 32 bit accumulator clocked at the decimated sampling rate, which is 500 kHz for the data presented here resulting in a frequency control LSB of 0.11 mHz.

The control block multiplexes the four gyroscope parameters to a pipeline of a first order 16 bit IIR filter and PID. The IIR filter and PID blocks each contain a multiplier and accumulator which are again multiplexed to perform the needed calculations. The pipeline takes 12 cycles to fully execute and a new value can be introduced every 4 cycles. Because of this, 24 cycles at the master clock rate ($0.375\mu s$) are required to process all of the controls. One output is produced for each input, so the total delay is one cycle at the decimated sampling rate.

To reduce resource consumption, the PID and filter operate on 16 bit inputs, although the multiplication operations in each block result in 32 bit outputs. Rounding the signals through a mask operation as

$$\text{short} = \text{long[31:16]}$$

results in an offset in the signal. Positive values smaller than 2^{16} are rounded to 0, while negative values greater than -2^{16} are rounded to -1. If this rounded signal is integrated, as in the PID controller, there is trend to negative values. To avoid this effect, the signals are rounded as

$$\text{short} = \text{long[31:16]} + \text{long[15]}$$

which effectively implements rounding. If *long* is considered to be a floating point number with 16 bits for the integer and 16 bits for the decimals, *long[15]* indicates the 2^{-1} power, or 0.5. If *long[15]* is true, the decimal portion must be greater than or equal to 0.5, and if it is false it is less then 0.5. This rounding results in all negative values greater than 0xffff0000 being rounded to 0, which removes the bias so that the integral of the rounded value does not grow when the input is unbiased.

Drive signals are generated by applying a 14bit delta-sigma modulation to the control signals and then modulating the signals with quadrature +1/-1 square waves.

I_x and I_y are modulated with the in-phase square wave, and Q_y with the out-of-phase (90° delayed) square wave. The modulated I_y and Q_y signals are then summed and the result is routed to the Y-axis DAC. The modulated I_x signal is routed to the X-axis DAC. Using square waves instead of generating sinusoidal signals reduces the firmware image by 15%, and does not seem to affect performance.

The FPGA also implements a feed-through cancellation (FTC) block. Differential sensing does not remove all of the drive signal feed-through due mismatch in the device and amplifiers. This feed-through limits the effectiveness of error compensation control loops. The characterization suite can measure the feed-through, and then the FTC block scales the control signals by the provided scale-factors and subtracts that value from the corresponding demodulated sense signal, accounting for the 90° phase shift of the feed-through signal. The residual feed-through with low frequency gyroscopes is very small and more testing with higher frequency gyroscopes is required to evaluate the effectiveness of the feed-through compensation.

4.2.5 Square versus Sine Wave Drive

Either a square wave or sine wave can be used to drive the gyroscope. In the rate mode, there is room in the FPGA for either drive, however to implement multiple controls in one FPGA, using the square drive greatly reduces the required resources. To use a square drive signal, it is necessary to use the bias method of applying drive signals, discussed in Section 2.2.2.

4.3 Rate Gyroscope Experimental Results

Many different gyroscopes are expected to be compatible with the presented controls and we are working on testing more gyroscopes. In order to compare gyroscopes across control systems, it is useful to define a scale-factor that is dependent only on the gyroscope. For open-loop operation, the rate can be extracted from rate-sense axis amplitude over the drive axis amplitude [46]

$$\frac{A_{rate}}{A_{drive}} = \frac{c_y}{c_x} = A_g \tau (\Omega + \Omega_{offset}) \tag{4.10}$$

For closed-loop operation, assuming the that the gains for the drive signals on the drive and rate-sense axes are equal, the ratio of the amplitude control and angle or rate control is

$$\frac{A_{FFB}}{A_{ampl}} = \frac{I_y}{I_x} = A_g \tau (\Omega + \Omega_{offset}) \tag{4.11}$$

where $\theta_{control}$ is the orientation (rate force-feedback) signal, $E_{control}$ is the vibration amplitude control signal, A_g is the gyroscope angular gain, τ is the sense axis damping time constant, Ω is any applied rotation and Ω_{offset} is the gyroscope zero rate offset. The rate scale-factor $A_g \tau$ should be independent of the readout circuit and is a function of gyroscope mode shapes and mode matching, making it a useful measure of the gyroscope performance. The scale-factor can be extracted from the slope of the fitted rate data as in Figure 4.7.

Rate performance of a CING gyroscope has been measured with the presented controls. The rate performance has been measured using the open-loop, open-loop with quadrature control, closed-loop, and closed-loop with quadrature control. The gyroscope used for these tests was a low-frequency CING gyroscope [35]. The tuning of the gyroscope is plotted in Figure 4.5, and the modes are matched within 100 mHz. The average damping time constant is 9 s (Q of 85,000). The temperature was not controlled.

The Allan Variance of the gyroscope in the four modes is plotted in Figure 4.6 and the scale-factor plots in Figure 4.7. Because the quadrature control stabilizes the frequency mismatch, the bias stability improves with the quadrature control enabled. The closed-loop control also reduces the scale-factor variation and further improves the bias stability. The performance with the four different configurations is

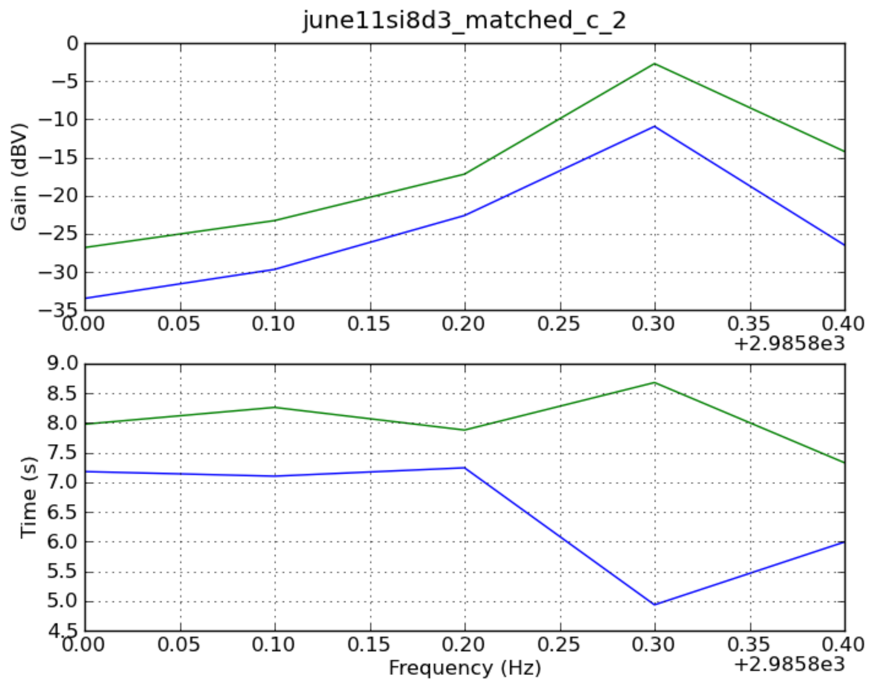


Figure 4.5: Mode response of gyroscope used for FPGA rate tests. The gyro was electrostatically tuned to have small mismatch, and then the tuning was adjusted while running in closed-loop to improve the sensitivity.

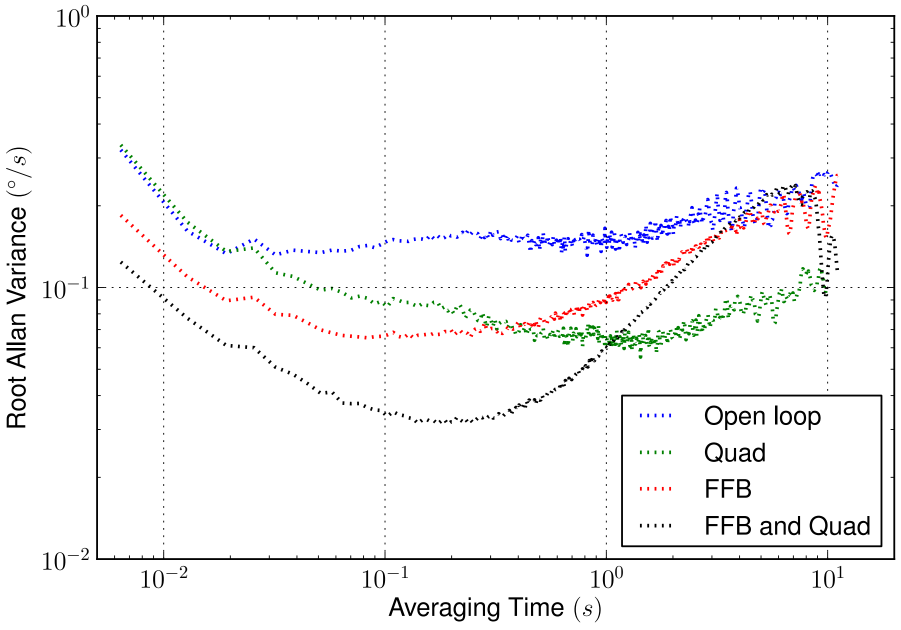


Figure 4.6: The feedback loops significantly increase the stability of the gyroscope resulting in improved AVAR plots. The noise performance is also improved since the sensitivity is improved with the feedback due to reduced effective frequency mismatch.

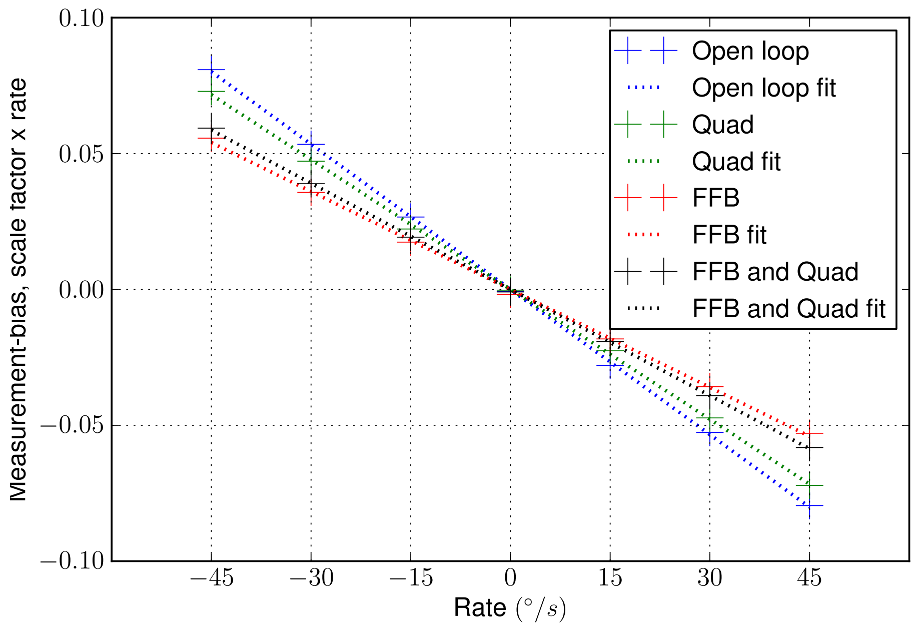


Figure 4.7: The measurement for open-loop modes is rate axis amplitude over drive axis amplitude, for closed-loop it is force feedback amplitude over drive amplitude. The scale-factor for each of these is $A_g\tau$ if the frequency mismatch is zero.

summarized in Table 4.1.

The USRP has an undesirable level of coupling between the DAC output and ADC input. This is especially pronounced on the second DAC channel with the LFRX board and may be due to the layout of the board. The results presented here use a custom interface board instead of the standard USRP LFTX and LFRX boards used in the characterization results. The custom board implements a differential second order filter and provides the fully differential connections to the USRP DACs and ADCs. The results are similar to results achieved with the standard boards.

4.4 Multiplexed Gyroscope Control

Time multiplexing of drive and sense signals for a single gyroscope has been previously suggested [51] for the extremely large damping time HRG. In that case however,

Table 4.1: Low-frequency CING rate performance with different controls. Stability of a low-frequency CING gyro when controlled with the FPGA rate control improves with force-feedback and quadrature control, but the angle random walk is not affected due to the small frequency mismatch.

Mode	Sensitivity $(^\circ/s)^-1$	Offset $(^\circ/s)$	ARW $(^\circ/\sqrt{Hr})$	Bias Stability $(^\circ/Hr)$
Open-loop	-561	-22.2	8.8	712
Quad Comp	-628	7.93	2.6	300
FFB	-833	16.25	2.3	353
Quad + FFB	-768	-1.16	0.8	170

only the drive and sense functions were multiplexed and the multiplexing was done per cycle with square drive signals. With multiple sensors, the phase and frequency will vary for each sensors and it would be difficult to arrange each sensors drive signal with a mux window. Also, a single square pulse will have a wide spectrum which can introduce noise in other sensors.

To reduce spectral leakage, a triangle shaped window is applied to both the drive signal and the received signal. Since the signal is not sensed continuously, filtering must be done carefully. A two stage filter is used. First a short duration filter to reduce the effect of harmonics while demodulating the sense signal which produces one output for each period when the channel is sensed. Second, a filter at the multiplexing frequency to smooth out any remaining discontinuities.

4.4.1 Multiplexing Circuit

The sense circuit for the multiplexed sensor module needs to settle quickly when the source is switched, which strongly influences the choice of sense amplifier. The drive circuit is illustrated in Figure 4.8 and the sense circuit in Figure 4.9. These are implemented with COTS chips, and integrating them into a single chip would greatly reduce the parasitic capacitance due to the switches and reduce the switching noise.

A CING gyroscope has been controlled using this method, with the control im-

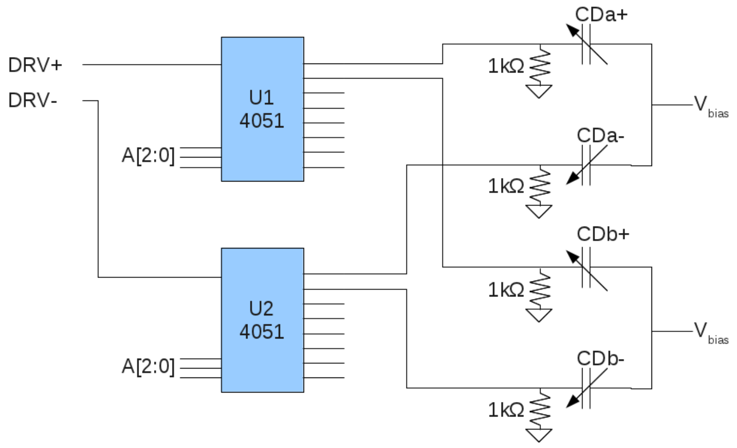


Figure 4.8: Diagram of the multiplexed drive circuit. The minimal drive circuit for a set of multiplexed sensors consists of an analog mux and resistors to ensure the bias voltage does not drive when the drive is disabled.

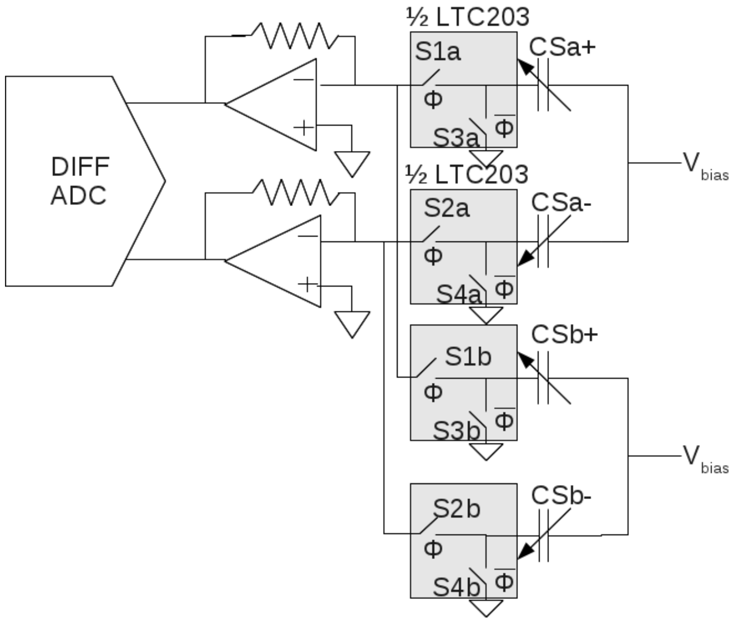


Figure 4.9: Diagram of the multiplexed sense circuit. The sense circuit for a set of multiplexed sensors must minimize charge injection to avoid disturbing the oscillation or saturating the sense amplifier.

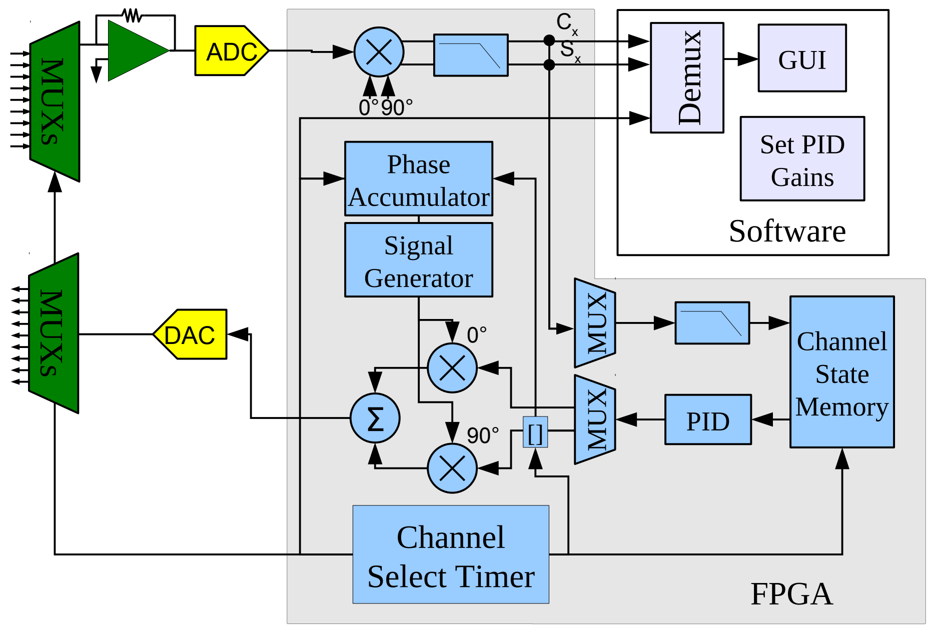


Figure 4.10: Diagram of the multiplexed control. This control is a modification of the single gyro rate control except with the control applied to multiple devices in turn. The state of the filters and PIDs in the control are saved for each device. The timing is controlled by a timing block which coordinates the drive and sense signals and insures break-before-make operation. The signal for each channel is reconstructed on the computer in real time by demultiplexing and filtering so that one value is produced for each time a channel is addressed.

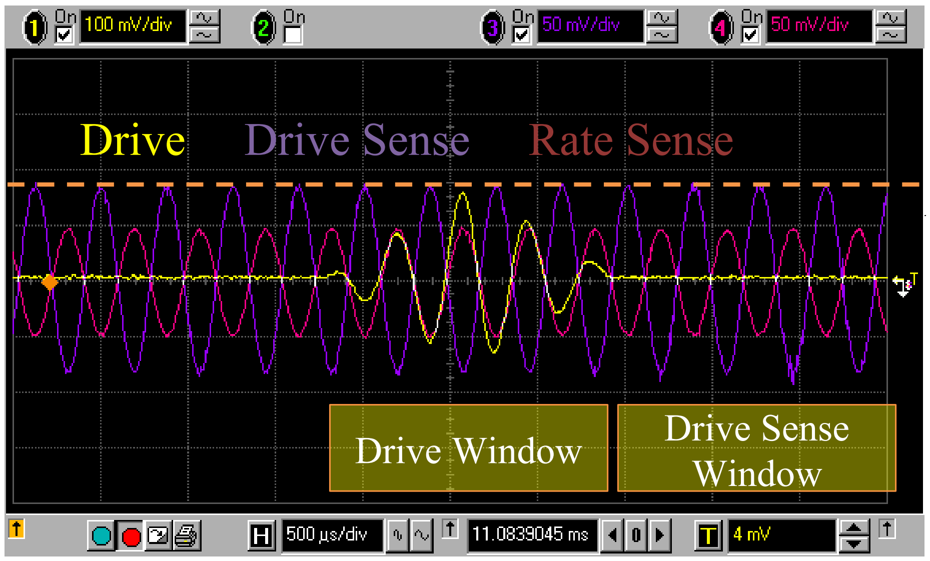


Figure 4.11: Demonstration of CING operating under multiplexed control. The control is multiplexed as if there were 8 channels present, such as 3 gyroscopes and 2 accelerometers, but the switches are not enabled to allow constant viewing of the sense signals.

plemented as shown in Figure 4.10. The rate performance has not been tested yet, however the amplitude is maintained and the phase does not drift noticeably during drive segments. The sense and drive signal on the CING are shown in Figure 4.11.

4.5 Summary

The control of a mode-matched CING gyroscope using the same SDR hardware as used for characterization has been demonstrated. The mode-matched performance improves with the addition of rate force-feedback and quadrature control. The FPGA is currently optimized to minimize resources through the use of pipelined controls, first order IIR filters, and square wave drive. Improving the filter is expected to improve performance. The efficient FPGA design means that several gyroscopes could be controlled by time-multiplexing the controls for several gyroscopes. A proof-of-concept for this approach was demonstrated showing that the phase and amplitude are maintained.

CHAPTER V

Rate Integrating Gyroscopes

A RIG is a gyroscope configured to mechanically integrate an applied rate of rotation and output a signal proportional to the total applied rotation. RIGs offer three primary advantages over rate gyroscopes: mechanically unlimited bandwidth, sensitivity that is purely a function of geometry, and wide dynamic range. The primary disadvantages of RIGs are the increased complexity of the controls and increased drift. While an ideal RIG will have zero drift, in practice even relatively small mismatch in the parameters of the degenerate modes will produce significant drift. Potential applications are miniature flying vehicles and personal navigation systems which could benefit from the very large bandwidth and dynamic range of RIGs.

The principle of a vibratory RIG is that the orientation of a standing wave on a structure will lag behind the rotation of that structure. The absolute rotation of the structure is measured by tracking the orientation of the standing wave with respect to the structure. Currently, RIGs rely on extreme manufacturing tolerance to achieve rate-integration. Traditional controls for a RIG track the oscillation phase, maintain total oscillation energy and compensate for quadrature error. An optional orientation control loop can set the standing wave orientation, and is typically used for rate-mode operation. In this chapter we present the modeling and simulation of RIG along with basic controls and the architectures used to implement them. Both simulation and

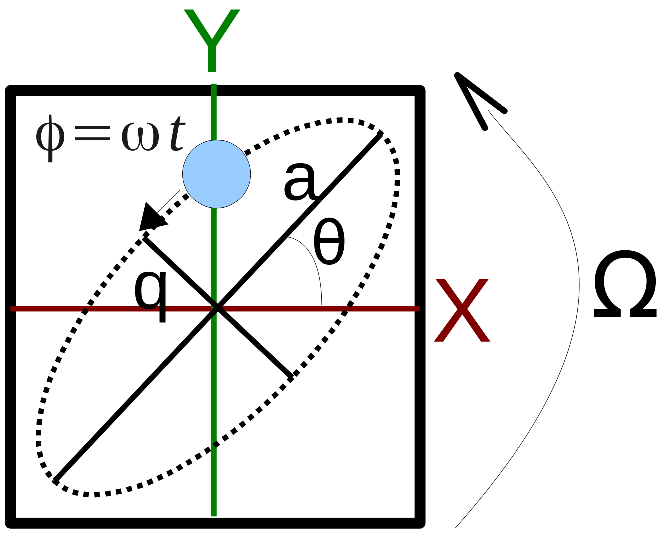


Figure 5.1: Diagram of a single mass Coriolis vibratory gyroscope. A single mass attached to a frame with springs aligned with the X and Y axis can be used as a generic model of all Coriolis vibrator gyroscopes.

measurement results are also presented.

5.1 Gyroscope Model

A simple conceptual vibratory gyroscope is illustrated in Figure 5.1. Unlike a rate gyroscope which has a drive axis and a rate-sense axis, a RIG has two equivalent axes which can be referred to as X and Y. A mass is vibrated with amplitude a and resonance frequency ω along an angle θ with respect to the X axis. Due to non-ideal features of any physical gyroscope, some quadrature motion q develops, which leads to drift in the orientation θ. Sensors are placed to measure the motion along the X and Y axes and the motion of the mass projected onto these axes is sensed as [19]

$$\begin{aligned} x &= acos(\theta)cos(\omega t) + qsin(\theta)sin(\omega t) \\ y &= asin(\theta)cos(\omega t) - qsin(\theta)sin(\omega t). \end{aligned} \tag{5.1}$$

Instead of a single mass, it is also possible to use any other structure with degenerate orthogonal modes such as the wineglass mode of a ring or cylindrical resonator [53].

The sensed gyroscope motion can be IQ demodulated as

$$\begin{bmatrix} c_x & c_y \\ s_x & s_y \end{bmatrix} = \begin{bmatrix} \cos\phi & \cos\phi \\ \sin\phi & \sin\phi \end{bmatrix} \begin{bmatrix} x_s \\ y_s \end{bmatrix}$$

where $\phi = \omega t + \phi'$ is the phase of the demodulating signal which may have some error ϕ'. The parameters c_x, c_y, s_x, s_y can be used to calculate the gyroscope control parameters [19]

$$\begin{aligned} E &= c_x^2 + s_x^2 + c_y^2 + s_y^2 = a^2 + q^2 \\ Q &= 2(c_x s_y - c_y s_x) = 2aq \\ R &= c_x^2 + s_x^2 - c_y^2 - s_y^2 = (a^2 - q^2)\cos 2\theta \\ S &= 2(c_x c_y + s_x s_y) = (a^2 - q^2)\sin 2\theta \\ L_i &= 2(c_x s_x + c_y s_y) = (a^2 - q^2)\sin 2\delta\phi \end{aligned} \tag{5.2}$$

where E is the oscillation energy, Q is the quadrature error, R and S are the X,Y projections used to calculate the oscillation angle, $\delta\phi$ is the phase error between the gyroscope oscillation and the reference signal, and L_i is the phase error in the demodulation signal.

5.2 Rate-Integrating Gyroscope Controls

The parameters E,Q,R,S, and L_i form the basis of the controls used in this work. A basic RIG control system is illustrated in Figure 5.2. There are four controls loops: amplitude, quadrature, orientation, and phase. In an ideal gyroscope, only the amplitude and phase control are necessary. The quadrature control cancels the quadrature error to reduce the drift due to variation in the resonance frequency and

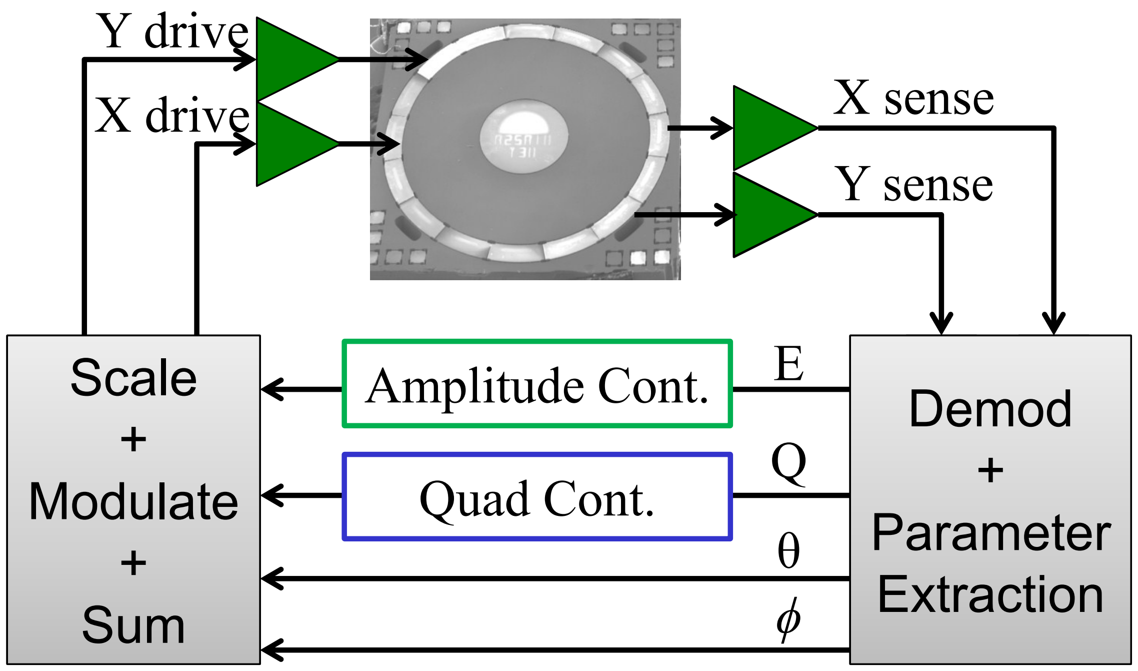


Figure 5.2: Basic diagram of a RIG control scheme. A RIG control system must implement blocks to maintain the total oscillation energy, cancel quadrature error, and proportion drive signals according to the oscillation orientation.

damping with the oscillation orientation. The orientation or angle control can steer the oscillation to a desired orientation. The output of the orientation control is proportional to the rate of rotation plus any intrinsic drift at that orientation. The amplitude, quadrature and orientation control values are converted to drive signals proportioned between the X and Y axes based on the measured value of the oscillation orientation θ.

The gyroscope is driven with drive signals which are the sum of in-phase (i) and out-of-phase (q) signals for each axis. The construction of i_x, q_x, i_y is

$$\begin{aligned}
i_x &= G_x(E_{pid}\cos\theta - S_{pid}\sin\theta)\\
i_y &= G_y(E_{pid}\sin\theta + S_{pid}\cos\theta)\\
q_x &= G_x(Q_{pid}\sin\theta)\\
q_y &= -G_y(Q_{pid}\cos\theta)
\end{aligned} \tag{5.3}$$

where E_{pid} is the output of the amplitude control PID which drives the energy measure E to the energy set point E_0, Q_{pid} is the output of the quadrature control control PID which drives Q to zero, and S_{pid} is the output of the rate or angle control control PID which drives θ to a set point θ_0. The gains G_x and G_y are used to correct for gain errors in the drive electronics.

The drive signals are then used to create modulated drive signals X_{drive} and Y_{drive} as

$$\begin{aligned} X_{drive} &= i_x C_{mod} + q_x S_{mod} \\ Y_{drive} &= i_y C_{mod} + q_y S_{mod} \end{aligned} \tag{5.4}$$

where C_{mod} and S_{mod} are the modulating signals which may be cosine and sine signals or in-phase and quadrature square wave signals.

The gyroscope orientation θ is often given as

$$\theta = \tan^{-1}(S/R)$$

however, this does not capture the full range of θ and it is not immediately clear how to extend this to $2\theta \in [-2\pi, 2\pi]$ required for the advanced controls presented in Chapter VI. The method used in our control is

$$2\theta = \tan^{-1}(S/R) + \begin{cases} 0, & \text{if } R > 0 \text{ and } c_x > 0 \\ -2\pi, & \text{if } R > 0 \text{ and } S > 0 \\ 2\pi, & \text{if } R > 0 \text{ and } S < 0 \\ -\pi, & \text{if } R \leq 0 \text{ and } \theta_{-1} < 0 \\ \pi, & \text{if } R \leq 0 \text{ and } \theta_{-1} > 0 \end{cases}$$

where θ_{-1} is the previous value of θ. Since R switches from positive to negative when θ is $\pm\frac{\pi}{2}$, θ_{-1} will not change sign when R is less than 0.

5.2.1 Mechanical Requirements

The mechanical requirements for the RIG control as presented here are similar to the requirements for mode-matched rate gyroscope control. In a RIG, the mechanical axes of the gyroscope are interchangeable, so the gain of the sense mechanisms should be the same for the two axes. This applies to the sense capacitance as well as the transducer and signal conditioning gain. Both axes of the gyroscope also need to be capable of being actuated and again the gains should be equal.

5.3 Orientation feed-forward

To reduce the effect of delay, which is quantified in the next section, the control system can measure the system delay and shift the control angle by

$$\theta_{tx} = \theta_{rx} + \Omega_{rx} t_{delay} \tag{5.5}$$

where θ_{tx} is the angle used to scale the drive signals, θ_{rx} is the measured angle, Ω_{rx} is the measured rate, and t_{delay} is the measured delay. The delay should be constant and is determined during an initial calibration step. In systems where the delay is not guaranteed to be constant, a side channel can be used to continuously update the delay.

5.4 Rate Integrating Performance

5.4.1 RIG Performance Metrics

The standard measures of performance in a rate gyroscope: ARW, bias stability, bandwidth, dynamic range, do not obviously translate into metrics for a RIG. Here we will describe the RIG analogs of these metrics and how they can be measured or derived.

A RIG measures the displacement on the X and Y axes and calculates the orientation as

$$\theta = \arctan \frac{y}{x} \tag{5.6}$$

where θ is the measured angle, y is the Y axis displacement and x is the X axis displacement. The angle noise of the gyroscope is then

$$\tilde{\theta} = \hat{\theta} - \theta = \arctan \frac{y + \tilde{y}}{x + \tilde{x}} - \theta \tag{5.7}$$

where $\tilde{\theta}$ is the measurement noise, $\hat{\theta}$ is the measured angle, and $\tilde{y}$,$\tilde{x}$ are the displacement measurement errors. This can be simplified based on the propagation of errors and a Taylor expansion of arctan, but the derivation is complicated. For a given oscillation amplitude r such that $x = r\cos\theta$ and $y = r\sin\theta$, the angle noise as a function of the displacement measurement noise assuming $\sigma(\tilde{x}) = \sigma(\tilde{y})$ is

$$\sigma\tilde{\theta} \approx \frac{\sigma\tilde{x}}{x^2 + y^2} = \frac{\sigma\tilde{r}}{r} \tag{5.8}$$

where σ indicates the standard deviation of the random variable. To verify this approximation, the simulated standard deviations of some angle measurements are plotted in Figure 5.4.1 for different SNR $(\sigma(\tilde{r})/r)$.

The angle random walk of a RIG is not simply the noise of the derivative of the angle, especially since the derivative of white noise is not white noise. The important characteristic of angle random walk is the effect it has on the measured angle over time. There is an angle random walk equivalent due to the use of the angle in drive loop of the gyroscope because angle noise creates an error in the feedback signal.

If the drive signals are

$$\begin{aligned} F_x &= E_c \cos(\hat{\theta}) \approx E_c \cos(\theta) - E_c\tilde{\theta}\sin(\theta) \\ F_y &= E_c \sin(\hat{\theta}) \approx E_c \sin(\theta) + E_c\tilde{\theta}\cos(\theta) \end{aligned} \tag{5.9}$$

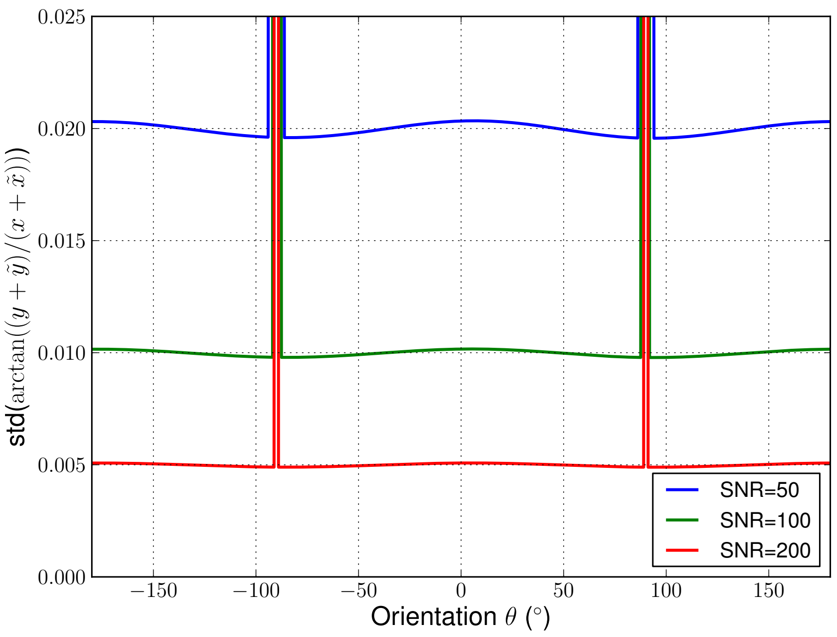


Figure 5.3: Standard deviation of an angle measurement. The standard deviation of an angle measurement is angle dependent, especially if the SNR is low. The discontinuities are due to the discontinuity of the arctan function around ±90°and can be removed with a more sophisticated calculation of arctan as in Equation 5.2.

these forces have the same form as the amplitude control plus orientation control if $E_c\tilde{\theta}$ is replaced with $\theta_{control}$. This error will then apply a rate to the gyroscope just as the orientation control does, and this rate will be integrated by the gyroscope and accumulate as an angle error in the same way as ARW. If the rate due to the angle noise is calculated, the applied rate due to angle measurement noise is

$$\Omega_{RIG,ARW} = \frac{\theta_{control}}{E_{control} A_g \tau} = \frac{E_{control}\tilde{\theta}}{E_{control} A_g \tau} = \frac{\tilde{\theta}}{A_g \tau} \tag{5.10}$$

It is also possible to write the angle random walk in terms of the displacement noise

$$\Omega_{RIG,ARW} = \frac{\sigma\tilde{r}}{r A_g \tau}. \tag{5.11}$$

If a rate gyroscope is operated along the X axis and the Y axis is used for sensing, then $\sigma\tilde{r} = \sigma\tilde{y}$ and $r = x$. It is clear then from Equation 4.1 and Equation 5.11 that for the same oscillation amplitude, the angle random walk is equal in rate and rate-integrating modes. The displacement noise can either be from mechanical noise or voltage noise, and so the RIG ARW is the same as the rate mode ARW independent of the displacement noise source.

Bias stability is not the same as gyroscope drift, since the drift can be removed if it is well known which makes it more like zero rate offset. The main difference is that the drift is angle dependent and symmetric so that if the gyroscope makes a full rotation, the drift is zero. The same environmental factors that change the zero-rate offset and contribute to bias instability will also lead to changes in the drift over time. This implies that the drift stability and bias stability will be roughly the same. However, if the gyroscope is rotated fast enough that it makes a full rotation in less time than is required to reach the bias stability by filtering the rate, the effective bias stability will be zero. This averaging time is generally from a few seconds for a low quality gyro to many hours for a high quality gyroscope with environment control.

The following rule for operating in RIG mode versus rate mode is proposed

$$\text{mode} = \begin{cases} \text{rate} & \text{if } 360/(\Omega A_g) > T_{stable} \\ \text{rate-integrating} & \text{if } 360/(\Omega A_g) \leq T_{stable} \end{cases}$$

Mismatch between the actual orientation and the orientation used to scale the control signal can also be created by delay in the control and this mismatch creates a rate-dependency in the scale factor of the gyroscope. Under high rates of rotation the calculated angle can significantly lag behind the actual angle. The amount of error caused by the delay can be analyzed in a similar way as the effect of angle noise.

The angle error due to delay τ_D is

$$\theta_{err} = -\Omega\tau_D$$

where the negative sign reflects the measured angle lags behind the actual angle. This error implies that the drive signal based on the measured angle $\hat{\theta} = \theta + theta_{err}$ is

$$F_x = E_c \cos(\hat{\theta}) \approx E_c \cos(\theta) - E_c\theta_{err} \sin(\theta)$$
$$F_y = E_c \sin(\hat{\theta}) \approx E_c \sin(\theta) + E_c\theta_{err} \cos(\theta)$$

which means that $E_c\theta_{err}$ looks like the rate control signal and the induced rate due to delay Ω_D in the gyro is

$$\Omega_D = \frac{E_{control}\theta_{err}}{E_{control}A_g\tau} = \frac{\theta_{err}}{A_g\tau} = \frac{-\Omega\tau_D}{A_g\tau} \tag{5.12}$$

and the measured rotation rate $\hat{\Omega}$ is the sum of the physical rotation and the rotation induced by the delay

$$\hat{\Omega} = \Omega + \Omega_D\frac{-\Omega\tau_D}{A_g\tau} = \Omega\left(1 - \frac{\tau_D}{A_g\tau}\right) \tag{5.13}$$

which is the actual rotation with a scale factor. The scale factor error is constant if the delay is constant, and contributes rate-random walk if the delay is not constant. The impact of the error decreases as the damping time constant of the gyroscope resonator is increased.

5.5 Software Control Architecture

The all-software control block diagram in Figure 5.4 consists of the gyroscope, analog interface circuitry, data transfer through the USRP, and all of the control blocks implemented in software in the GnuRadio architecture. For speed, custom low level processing blocks were used for the PLL, parameter extraction and PID. The gyro model PLL block consists of a PID to minimize the phase error signal ψ_e calculated according to [19]. The internal phase accumulator is used to generate sine and cosine signals for demodulating the incoming signals in the parameter extraction block. The PLL also takes a delay parameter which is multiplied by the instantaneous frequency to generate a phase delay which is added to the phase accumulator used to generate a second set of periodic signals used to modulate the output of the control blocks and generate drive signals. This arbitrary delay function is necessary to maintain the gyroscope oscillation close to the resonant frequency, which requires the total loop phase-shift is a multiple of 2π. The difficultly in achieving the right phase-shift when the delay changes each time the software is run is the primary drawback of the all-software control.

The most significant difference between the RIG PLL in Figure 5.5, and a normal PLL is the phase detector, which must tolerate the quadrature signal and the orientation of the oscillation. The phase error is calculated as L_i in Equation 5.2, The non-delayed output of the PLL is used by the parameter extraction block to demodulate the received signals. The demodulation accounts for any known gain mismatch and whether the signal is in phase with the oscillation velocity (for transimpedance

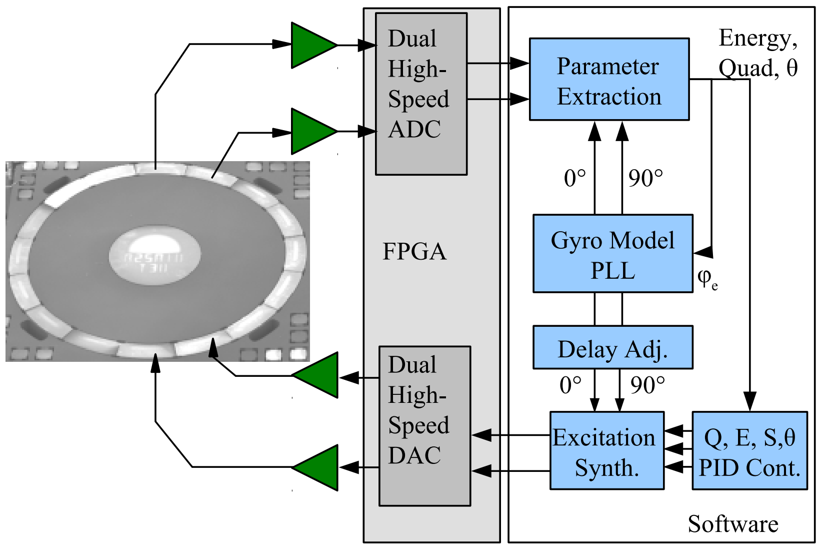


Figure 5.4: Diagram of the software control architecture. In software control mode, all of the controls are implemented in software. The FPGA is still used to decimate the signals so that the data rates fit within the USB bandwidth. The FPGA code has been modified from the standard firmware to prevent accumulating delay when the transmit buffer underruns due to the lack of a real-time operating system.

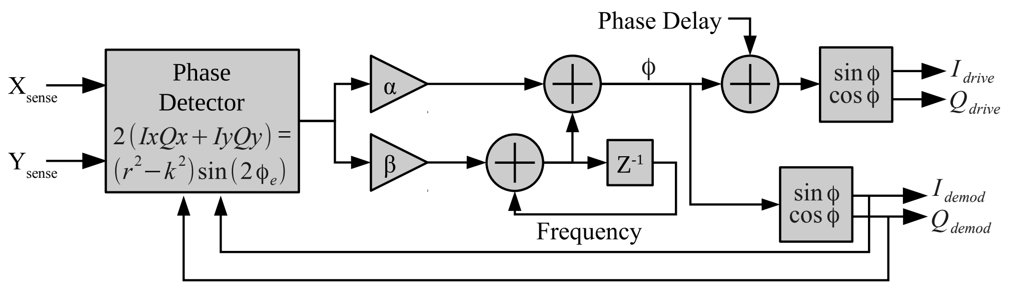


Figure 5.5: Diagram of a PLL with variable delay. One implementation of a PLL for a RIG. The delay in the PLL between the modulating and demodulating references signals can be adjusted to compensate for the uncertain delay due to the USB bus and buffer.

amplifiers), or the oscillation position (for charge-sense amplifiers). The demodulated signals are combined according to Equation 5.2 to form the energy level E, quadrature error Q, and orientation θ signals in the parameter extraction block. Each of these is fed to a proportional, integral, differential (PID) control. Both parameter extraction and PID control are implemented in custom blocks to reduce system load. The orientation block can also be used in the rate-integrating mode start-up procedure or for calibration. The control blocks are enabled in the order of PLL, amplitude control, quadrature control and then orientation control. The parameter extraction equations make assumptions about the control state which are not necessarily true during start-up, so stability and speed of start-up is improved if the blocks are started after the previous block has locked or nearly locked.

5.6 Hybrid Control Architecture

The hybrid control architecture in Figure 5.6 addresses the limitations of both the FPGA control and software control. Like the FPGA control, demodulation, modulation and phase control are implemented in the FPGA. These operations are very sensitive to phase and so the low, fixed delay in the FPGA is critical to good performance. The other control functions such as amplitude control and angle control

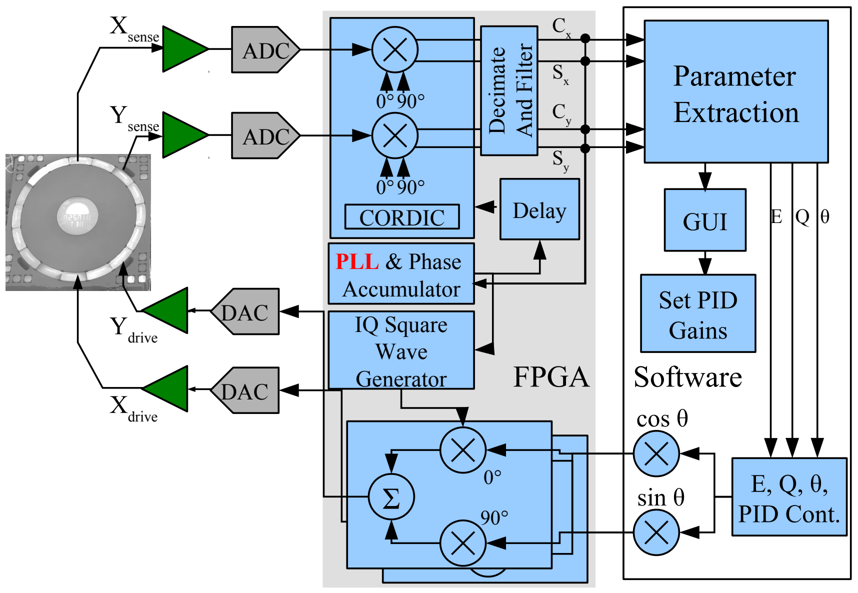


Figure 5.6: Diagram of the relay control architecture. In the relay mode architecture, the most delay critical blocks (PLL and feed-through cancellation) are implemented in the FPGA while the rest of the control is implemented in software.

are implemented in software. These controls need delays that are much smaller than the time constant of the gyroscope in order to be stable, but high-Q MEMS gyroscopes often have ring-down times tens of milliseconds and some like the CING are several seconds or more. The delay of the software control is a few milliseconds and so these controls can be stable. Implementing the controls in software allows for much more complicated controls to be implemented and for rapid implementation of new controls when developing algorithms.

In the FPGA, the PLL is a fixed point and pipelined implementation of Figure 5.5 using the same blocks as the controls in the FPGA rate mode plus an additional block to calculate L_i.

The software side of the hybrid control is similar to the all software control without the demodulation, PLL, and modulation functions. Instead c_x, s_x, c_y, and s_y are received from the FPGA, filtered, and used to calculate the control values in Equation

5.2. Software PIDs are applied to each control and the outputs of those PIDs are used to construct i_x, q_x, i_y, and q_y as in Equation 5.3 which are transferred back to the FPGA to be modulated.

5.7 Gyroscope Simulation

The simulation architecture illustrated in Figure 5.7 is built upon the control software we previously presented in [54],[35],[55]. On a fast laptop, the following simulation runs at about $\frac{1}{2}$ real time (10 seconds of simulated gyroscope dynamics and control requires roughly 20 seconds of real time) to twice real-time depending on whether 64 bit or 32 bit libraries are used. The USRP FPGA hardware is replaced with a software model of the PLL implemented in the FPGA, including the use of a first order IIR filter, with precision limited to 16 bits. The simulated PLL and the actual implementation in the FPGA are critical to the control performance. The reference phase is accumulated as a 32 bit integer at the same sampling frequency as used in the FPGA, 500 kS/s for these simulations, according to

$$\phi = \sum \omega_0 + \phi_{pll}.$$

The full gyroscope model is illustrated in Figure 5.8, and includes capacitive feed-through, misalignment of the drive and sense electrodes modeled as a rotation, and drive and sense gain errors. Non-linearity of the parallel plate drive and the effects of the bias voltage on the drive and sense signals are not currently modeled. Bias voltages used to modify the stiffness matrix are modeled however so that automatic tuning and the effect of miss-tuning can be modeled.

Capacitive feed-through is modeled by taking the derivative of the drive signals

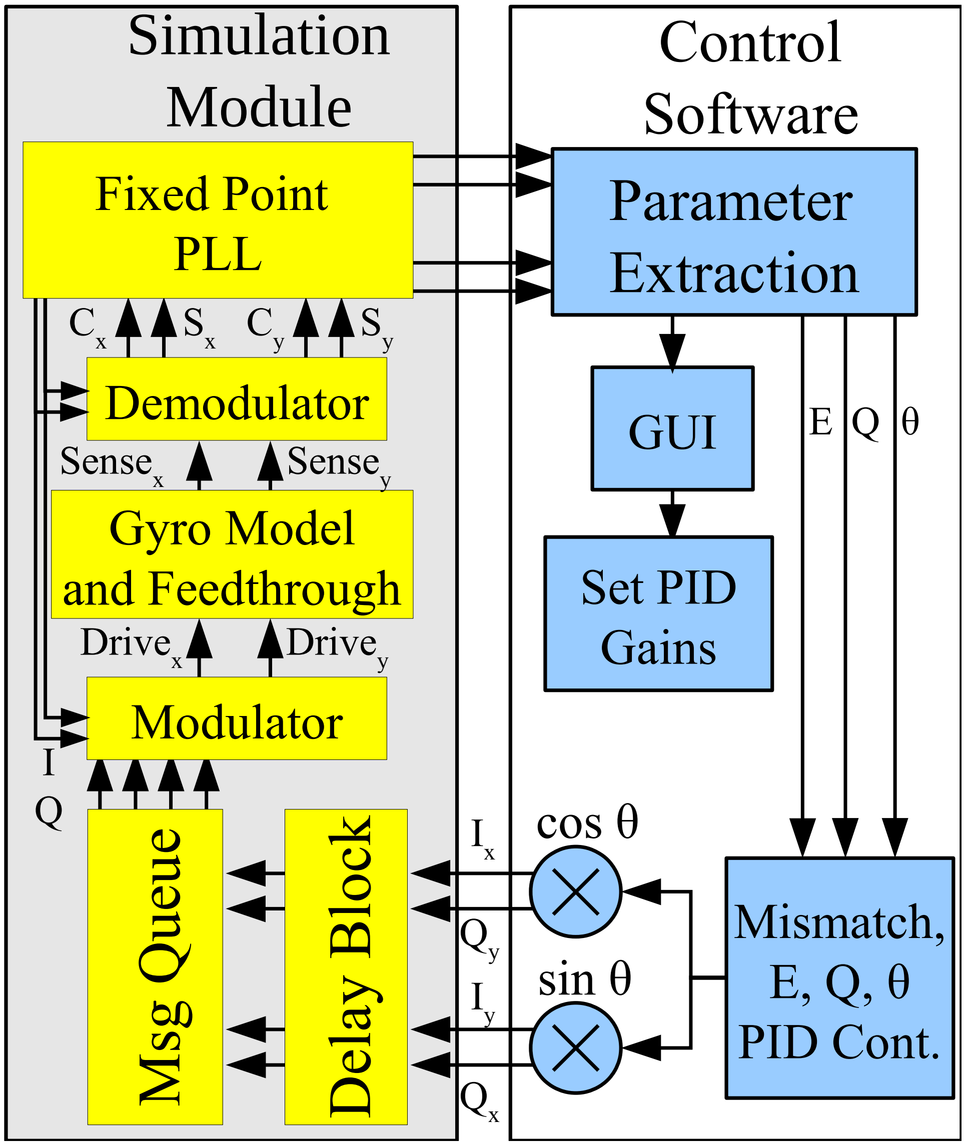


Figure 5.7: Simulation and Control Architecture. The simulation architecture used to test the proposed control functions. The simulation block replicates the FPGA firmware and gyroscope. The controls use the same code whether the software is being run in simulation mode or with physical hardware. Simulation mode is selected using a command line argument to the control interface software.

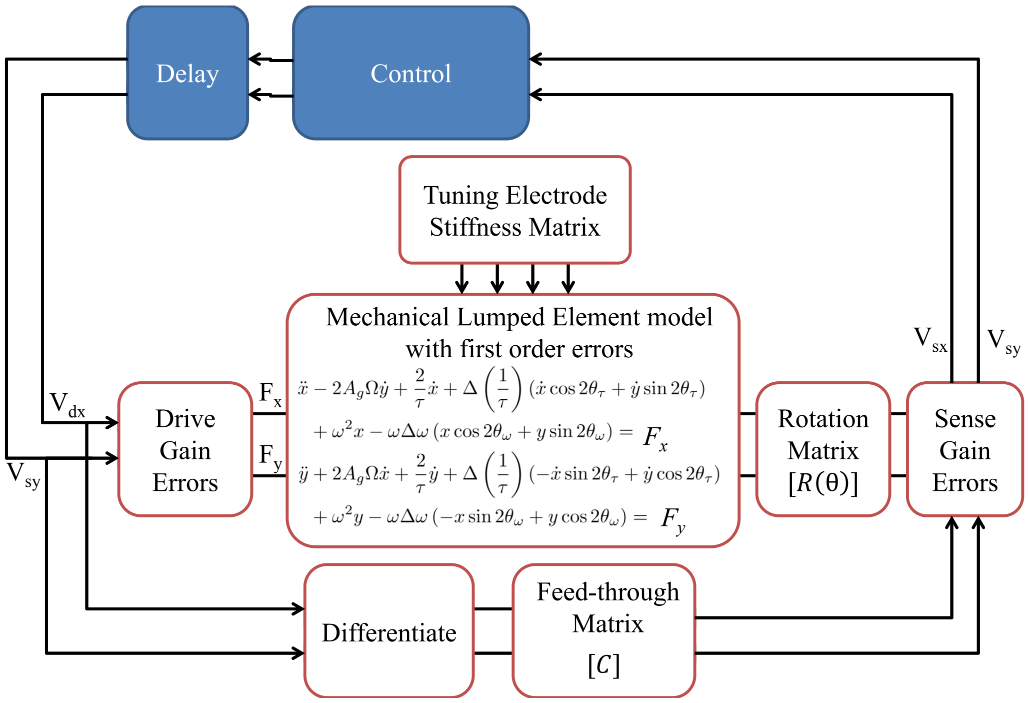

Delay
Control
Tuning Electrode Stiffness Matrix
Mechanical Lumped Element model with first order errors
$\ddot{x} - 2A_g\Omega\dot{y} + \frac{2}{\tau}\dot{x} + \Delta\left(\frac{1}{\tau}\right)(\dot{x}\cos 2\theta_\tau + \dot{y}\sin 2\theta_\tau)$
$+\omega^2 x - \omega\Delta\omega\,(x\cos 2\theta_\omega + y\sin 2\theta_\omega) = F_x$
$\ddot{y} + 2A_g\Omega\dot{x} + \frac{2}{\tau}\dot{y} + \Delta\left(\frac{1}{\tau}\right)(-\dot{x}\sin 2\theta_\tau + \dot{y}\cos 2\theta_\tau)$
$+\omega^2 y - \omega\Delta\omega\,(-x\sin 2\theta_\omega + y\cos 2\theta_\omega) = F_y$
V_{dx}
V_{sy}
Drive Gain Errors
F_x
F_y
Rotation Matrix $[R(\theta)]$
Sense Gain Errors
V_{sx}
V_{sy}
Differentiate
Feed-through Matrix $[C]$

Figure 5.8: Gyroscope model flowchart. The heart of the gyroscope model is Equation 2.2, however many electrical non-idealities are not included in those equations. The complete model used here includes gain errors, rotation effects, and feed-through. Modifying these parameters is an important tool in trying to identify the causes of behavior observed when controlling real gyroscopes.

and adding them to the output of the gyroscope model according to

$$\begin{bmatrix} x_{ft} \\ y_{ft} \end{bmatrix} = \begin{bmatrix} C_{xx} & C_{yx} \\ C_{xy} & C_{yy} \end{bmatrix} \begin{bmatrix} dx_d/dt \\ dy_d/dt \end{bmatrix}$$

where x_{ft}, y_{ft} are the feed through signals, x_d, y_d are the X-axis and Y-axis drive signals, and C_{uu} are coefficients chosen to match the measured feed through of MEMS gyroscopes tested in our lab. For the low frequency 3kHz CING, the feed-through is small enough to be ignored. It is more significant for the 20kHz CING.

The drive amplitudes provided by the control software are modulated by the output of the PLL as

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} I_x & Q_x \\ I_y & Q_y \end{bmatrix} \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix}$$

The modulated drive signals are used as inputs to the gyroscope simulation. The gyroscope simulation is a C++ implementation using the GSL [56] ODE solver of the equation 6 from [19] plus electronic tuning stiffness terms

$$\begin{aligned}
&\ddot{x} - 2A_g\Omega\dot{y} - A_g\dot{\Omega}y + \frac{2}{\tau}\dot{x} + \Delta\left(\frac{1}{\tau}\right)(\dot{x}\cos 2\theta_\tau + \dot{y}\sin 2\theta_\tau) \\
&\quad + \left(\omega^2 - A_g'\Omega^2\right)x - \omega\Delta\omega\left(x\cos 2\theta_\omega + y\sin 2\theta_\omega\right) + K_{el,xx}y + K_{el,yx}y = x_d \\
&\ddot{y} + 2A_g\Omega\dot{x} + A_g\dot{\Omega}x + \frac{2}{\tau}\dot{y} - \Delta\left(\frac{1}{\tau}\right)(\dot{x}\sin 2\theta_\tau - \dot{y}\cos 2\theta_\tau) \\
&\quad + \left(\omega^2 - A_g'\Omega^2\right)y + \omega\Delta\omega\left(x\sin 2\theta_\omega - y\cos 2\theta_\omega\right) + K_{el,yy}y + K_{el,xy}x = y_d
\end{aligned}$$

which includes angular acceleration and centripetal force as additional terms for quadrature error generation. The constant A_g is a function of the gyroscope geometry. The velocity variables $\dot{x}$,$\dot{y}$ are used as the output of the gyroscope simulation. The position variables can also be used as the output to model different sense amplifiers. The stiffness matrix from the tuning electrode model in Equation 3.3 are $K_{el,xx}$, $K_{el,yx}$, $K_{el,xy}$, $K_{el,yy}$, in the above model and are in parallel with the Ω^2 stiffness terms.

The gyroscope output is demodulated to create the gyroscope parameters used by the controls according to

$$\begin{bmatrix} c_x & c_y \\ s_x & s_y \end{bmatrix} = \begin{bmatrix} \cos\phi + \phi_{dly} & \cos\phi + \phi_{dly} \\ \sin\phi + \phi_{dly} & \sin\phi + \phi_{dly} \end{bmatrix} \begin{bmatrix} x_s \\ y_s \end{bmatrix}$$

where ϕ_{pll} is the output of the phase accumulator and ϕ_{dly} is ϕ_{pll} plus a constant to compensate or model any phase-shift in the circuits.

The control parameter L_i is calculated within the FPGA simulation and used as the error signal for the PLL PID. The demodulated sense signals are transferred to the control software where they are filtered and decimated and then used to calculate the control variables in (5.2).

5.8 Simulation Results

The above compensation schemes have been tested over a range of gyroscope parameters. The results presented below are for a gyroscope with a resonance frequency of 5 kHz, Q of 25,000 ($\tau = 1.6sec$) and A_g of 0.3. This A_g is predicted for a ring or hemispherical resonator from FEM simulation and is larger than the measured A_g for the CING resonator used in the experimental results section [35].

The drift due to frequency mismatch is proportional to the quadrature signal, which is effectively canceled by the quadrature control loop. In order to evaluate the reduction in frequency mismatch, it is useful to look at the rate of growth of the quadrature signal [15]

$$\dot{q} = \frac{q}{\tau} + \frac{a}{4\omega}(\omega_1^2 - \omega_2^2)\sin 2(\theta - \theta_\omega) \tag{5.14}$$

where a is the oscillation amplitude, q is the quadrature amplitude, ω is the average resonance frequency and ω_1 and ω_2 are the resonance frequencies along the principal axes, which are rotated θ_ω from the X-axis. The quadrature signal is maximum when $(\theta - \theta_\omega) = \pi/4$. To evaluate the residual frequency mismatch, at the steady state with $(\theta - \theta_\omega) = \pi/4$ and $\dot{q} = 0$

$$\frac{q}{\tau} = -\frac{a}{4\omega}(\omega_1^2 - \omega_2^2) \tag{5.15}$$

which can be solved for using the gyroscope parameters Q and E from Equations 5.2 as

$$\frac{Q}{E} = \frac{2aq}{a^2 + q^2} \approx \frac{2q}{a} = -\frac{\tau}{2\omega}(\omega_1^2 - \omega_2^2) \tag{5.16}$$

and rearranging with $(\omega_1^2 - \omega_2^2) = 2\omega(\omega_1 - \omega_2)$, the frequency mismatch is

$$(f_2 - f_1) = \frac{Q}{E 2\pi\tau}. \tag{5.17}$$

The residual effective mismatch calculated with Equation 5.17 for a gyroscope with τ of 10 s and quadrature compensation is plotted in Figure 5.9.

The steady state value of q is not reached if the gyroscope is rotating. Rotation modulates the frequency mismatch (this is easy to see if θ in Equation 5.14 is replaced with Ωt) and the integration to get q is a low pass filter, so a rotated gyroscope will not develop the full quadrature signal. The equations for calculating the angle and energy break down when $q > a$, which provides a limit on the mismatch for uncontrolled operation at slow rotation of

$$|f_1 - f_2| < (\pi\tau)^-1 \tag{5.18}$$

which is 32 mHz for τ equal to 10 s. With quadrature control, a gyroscope with only 32 mHz frequency mismatch would have very little drift. The limit of compensation for quadrature control comes from the relative size of the quadrature and drive signals

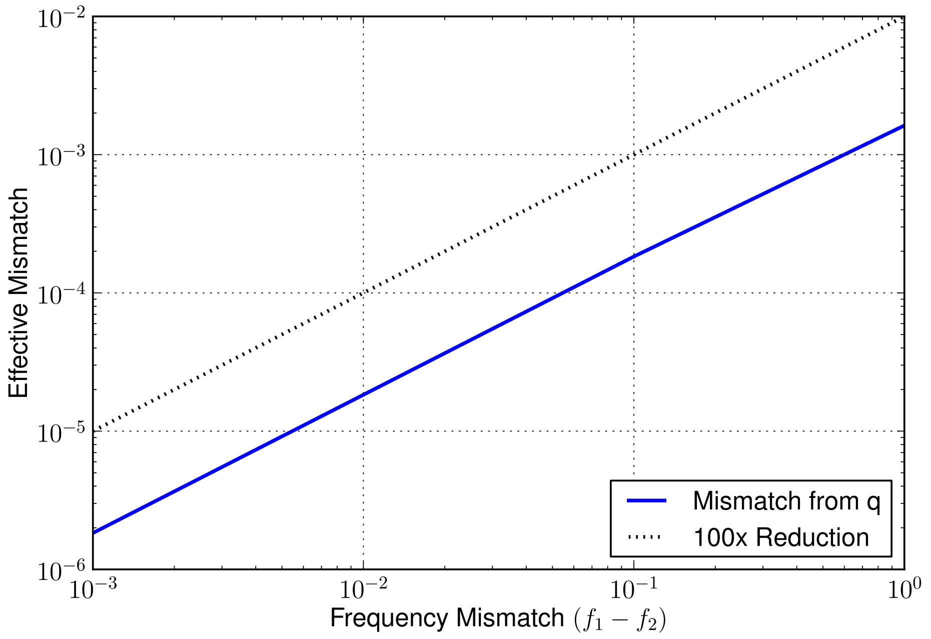


Figure 5.9: Reduction in effective frequency mismatch with quadrature control. Quadrature control effectively reduces the effective frequency mismatch by about three orders of magnitude in simulation.

in the same way as for rate gyroscopes as explained in Section 4.2.

To determine the drift versus angle in each simulation, a constant rotation rate is applied to the gyroscope ($60°/s$ or $90°/s$ for the largest $\Delta\tau$) and then the measured rate of rotation is calculated. The drift is the measured rate minus the applied rate. Because of delay in the control, which is modeled to represent delay in the real system, the drive signals lag behind the actual rotation and shift the average rate away from the applied rate, resulting in a slight positive bias in the drift plots.

The drift for several values of $\Delta\frac{1}{\tau}$ and $\omega\Delta\omega$ without the proposed mismatch controls are plotted in Figure 5.10. By the standard analysis, reducing the quadrature error to zero will reduce the corresponding drift to zero, however there is a residual drift proportional to the frequency mismatch that is an order of magnitude larger than predicted by the residual quadrature error. This drift is due to the effective damping mismatch introduced by the frequency mismatch since damping is proportional to

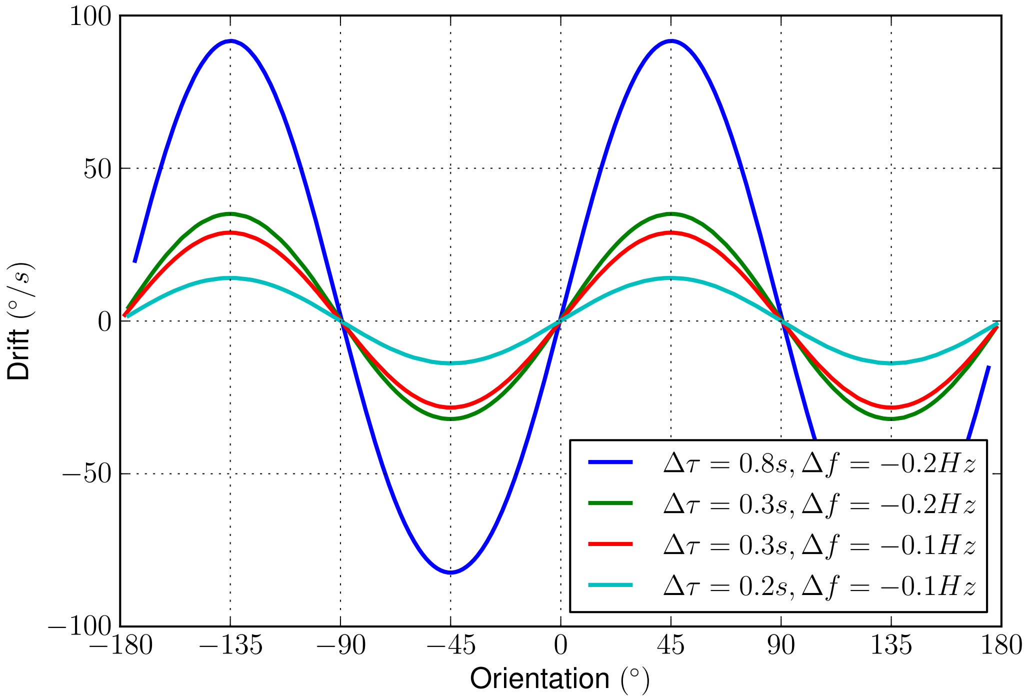


Figure 5.10: Simulated drift versus oscillation orientation with only quadrature compensation. The simulated gyroscopes has a nominal frequency of 5 kHz and Q of 25,000 ($\tau = 1.6sec$) with various amounts of mismatch.

$\omega(\theta)a/\tau$.

5.9 Measurement Results

A single program called gyrocontrol provides the interface and control for FPGA mode, software mode and hybrid mode control. A flow-chart of testing a gyroscope with the proposed system is illustrated in Figure 5.9. At the beginning of each test, the mode frequencies are matched within a fraction of a Hertz with electrostatic tuning. For a CING, the tuning voltages are under the gyroscope mass. The bias voltage for this CING gyroscope is 10V, and the tuning voltages are approximately -3V ($V_{bias} - V_{xtune} = 13V$) for cross-axis tuning and -1V ($V_{bias} - V_{ftune} = 11V$) for on-axis tuning. Simulation of the tuning mechanisms does not show any interference with the control system, however there is some concern that the large tuning voltages

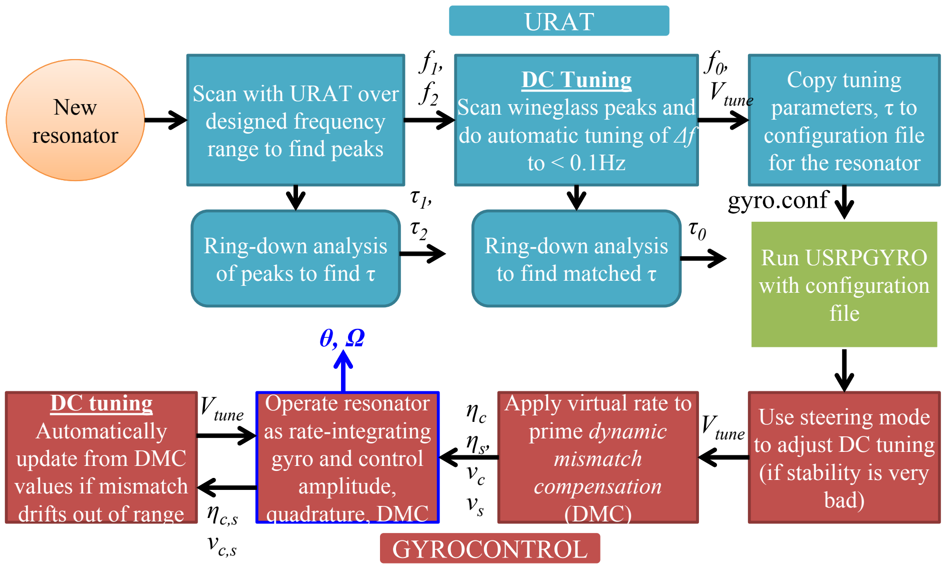

URAT
New resonator
Scan with URAT over designed frequency range to find peaks
f_1, f_2
DC Tuning
Scan wineglass peaks and do automatic tuning of Δf to < 0.1Hz
f_0, V_{tune}
Copy tuning parameters, τ to configuration file for the resonator
τ_1, τ_2
Ring-down analysis of peaks to find τ
Ring-down analysis to find matched τ
τ_0
gyro.conf
Run USRPGYRO with configuration file
θ, Ω
DC tuning
Automatically update from DMC values if mismatch drifts out of range
V_{tune}
$\eta_{c,s}$
$v_{c,s}$
Operate resonator as rate-integrating gyro and control amplitude, quadrature, DMC
η_c
η_s,
v_c
v_s
Apply virtual rate to prime dynamic mismatch compensation (DMC)
V_{tune}
Use steering mode to adjust DC tuning (if stability is very bad)
GYROCONTROL

Figure 5.11: Flowchart of operation of a gyroscope using URAT and gyrocontrol. The gyroscope is first tested and matched using URAT and then operated as a gyroscope using gyrocontrol, which can operate in rate or rate-integrating modes. Automatic closed loop tuning is only tested in simulation at this point, and several steps would benefit from automation.

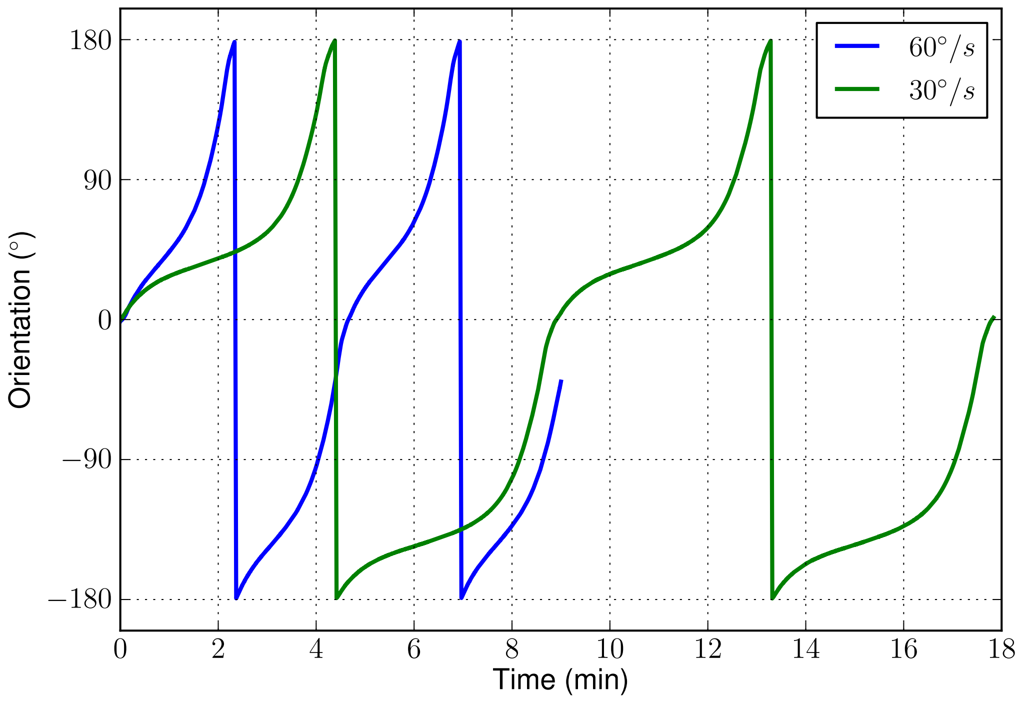


Figure 5.12: Angle measurement using rate-integrating control with constant applied rates. The Y-axis in this plot is the angle in the absolute coordinates of the CING (so Y is 45°). The rates were applied consecutively and then plotted with 0 minutes as the first zero crossing at each rate. Ideally the output would be a straight line, modulus 90° with a slope equal to the applied rate times A_g. The curve in the line represents the drift rate due to anisotropy.

might stress or unbalance the gyroscope in unknown ways. The following results are for control of low-frequency CING gyroscopes [35].

The relay control is capable of operating a CING as a RIG indefinitely, and has been demonstrated to operate continuously for hours at a time over several days.

To verify that the angle control measures and controls the angle correctly, an oscilloscope was used to verify that the drive and sense signal ratios match the values reported in software in both freely oscillating mode and with the angle control enabled. An example oscilloscope capture is shown in Figure 5.9. The sense signals are kept in phase by the quadrature control loop. Using markers, the angle calculated from the arctangent of the X and Y amplitudes is -33.69°and from the X and Y drive

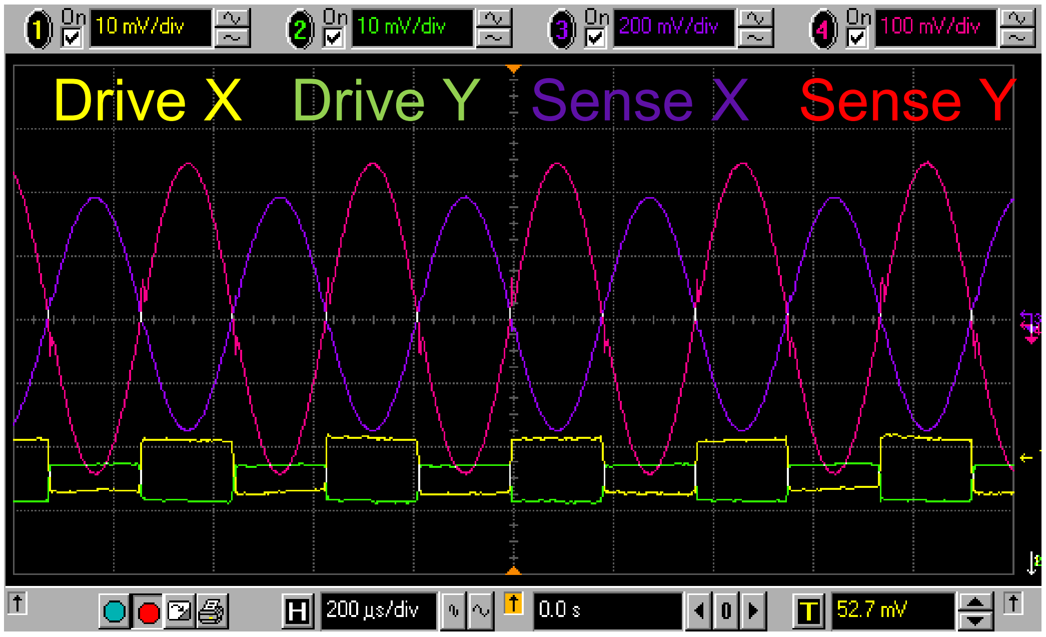


Figure 5.13: Drive and sense signals during angle-mode control measured with an oscilloscope. Measurements like this are used to verify that the angle control is behaving as expected.

amplitudes is -33.73°. The angle reported by the control is -33.78°. The angle has also been verified this way at 0°, 45°and 90°.

To verify the control accuracy, the approximate oscillation amplitude was calculated at different orientations as $\sqrt{V_x^2 + V_y^2}$ where V_x is the X-axis sense amplitude and V_y is the Y-axis sense amplitude. The measurement results are shown in Figure 5.9. The measured variation is up to 1%, but the oscilloscope introduces significant error making accurate measurements of the amplitude with the oscilloscope is difficult and that variance probably dominates the measurement error. The amplitude and quadrature control loops combine to keep this value constant. The nearly constant value against angle indicates that the control loops work and correctly combine the sensed signals.

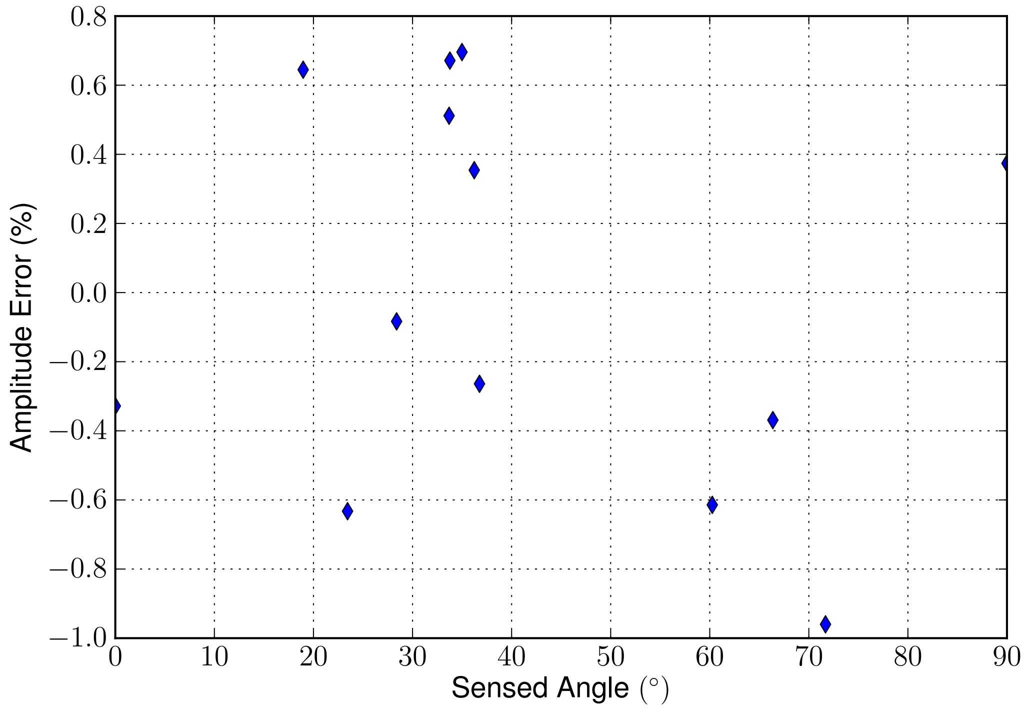


Figure 5.14: Variation in the energy signal against oscillation orientation. The energy is calculated by squaring the sensed voltages. The error is a combination of the oscilloscope channel mismatch, oscilloscope measurement error, monitor amplifier gain mismatch (the signal is amplified for the scope), and USRP errors.

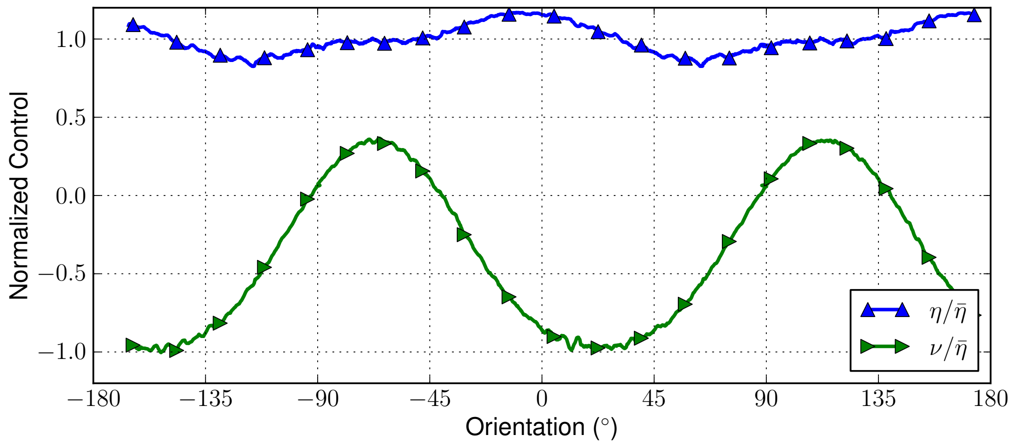


Figure 5.15: Frequency and damping variation measured using the quadrature and amplitude controls of a CING gyroscope. The wave orientation is controlled with electronic steering.

5.9.1 Model Verification

The control system presented here can be used to help verify and improve the gyroscope model. For a RIG, the dependency of the quadrature and damping signals is a function of the orientation. The normalized quadrature control and amplitude control signals are plotted in Figure 5.15. The expected shape for each curve is a shifted version of $\cos 2\theta$, however there are both large and small variations from that expected shape. The damping versus orientation for three consecutive sweeps of the angle are plotted in Figure 5.16 along with the expected $\cos 2\theta$ curve. In this plot there is a clear discrepancy between the expected damping around -20° and 70°. There are also many small excursions that look like noise, except that the excursions repeat with each pass. The large deviations are expected to be due to non-linearity in the drive signal or the structure.

The non-linearity can be identified by looking at the harmonics of the various control signals. The harmonic analysis is made by re-sampling the signal on each degree of orientation (so that there are 361 points) and then taking the FFT. The result is the components of the parameter at each wave number (wn). The wave

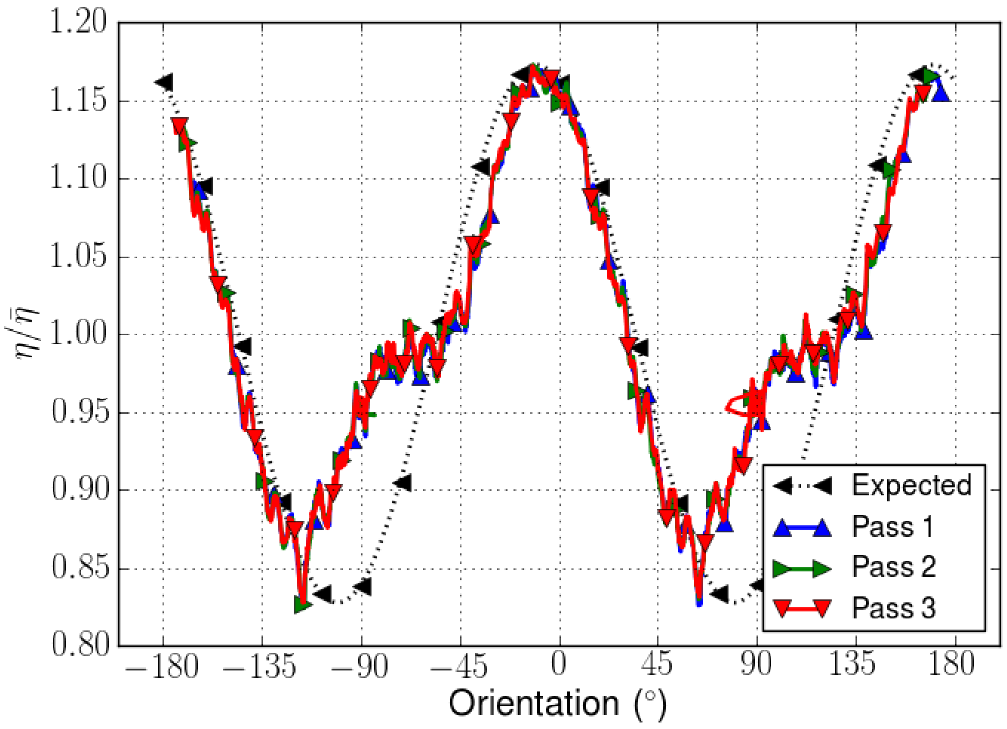


Figure 5.16: Measured damping mismatch in a CING gyroscope. The damping is derived from the variation in the amplitude control over 3 full rotations of the gyroscope oscillation.

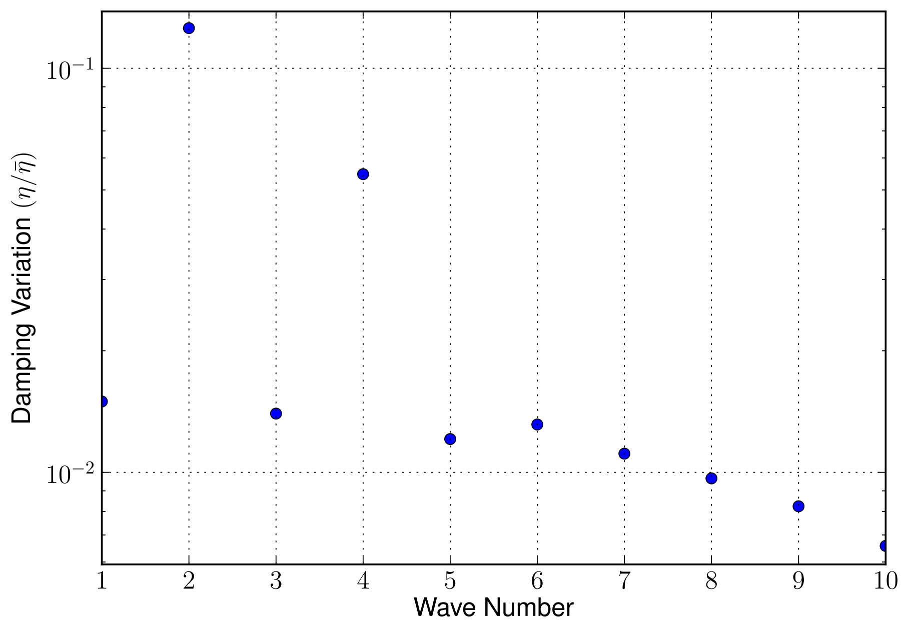


Figure 5.17: Harmonic analysis of the amplitude control signal. The amplitude control is proportional to the damping and shows large components at $\cos 2\theta$ and $\cos 4\theta$.

number is the number of times the signal repeats within 360°. So a wave number of 1 is be a dependency on $\cos\theta$, 2 is $\cos 2\theta$ which is the expected mismatch dependency, and 3 is $\cos 3\theta$ and so on. The harmonic analysis of the damping is plotted in Figure 5.17 and the harmonic analysis of the drift is plotted in Figure 5.18. Although the damping has a large component with wn=4, the drift is almost entirely at wn=2. This implies that the non-linearity of the damping is a function of the drive signal and not the mechanical structure. This implies that the dynamic damping mismatch control may need to use higher order components for the amplitude compensation but not the drift compensation.

The harmonic analysis of the quadrature control is plotted in Figure 5.19 and the harmonic analysis of the frequency is plotted in Figure 5.20. The quadrature does have some dependency on the damping as well as the frequency mismatch which may explain the difference in the harmonics between the quadrature and frequency.

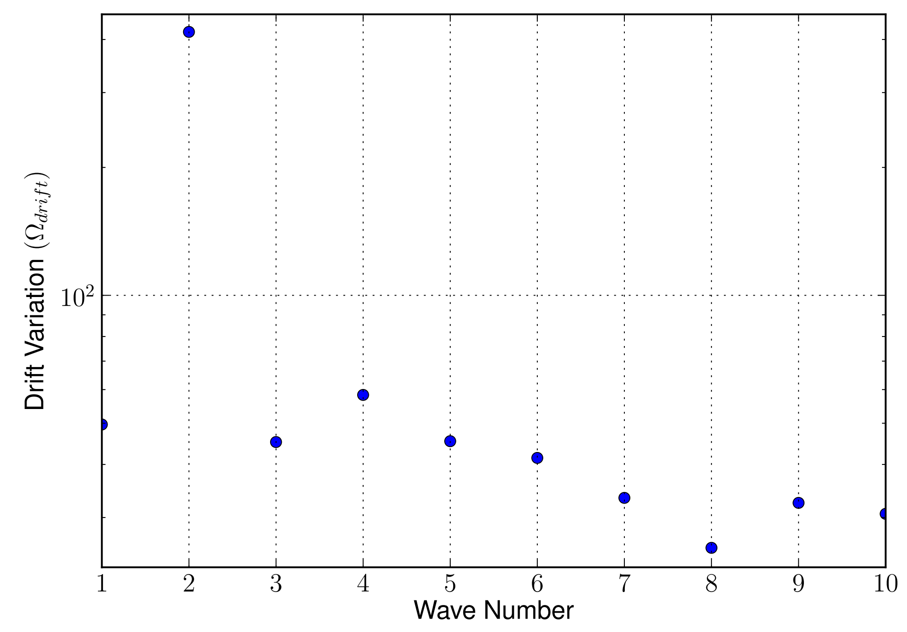


Figure 5.18: Harmonic analysis of drift in a CING gyroscope. Drift is expected to be dependent on the damping and frequency mismatch, and shows a large component at $\cos 2\theta$ and a much smaller one at $\cos 4\theta$.

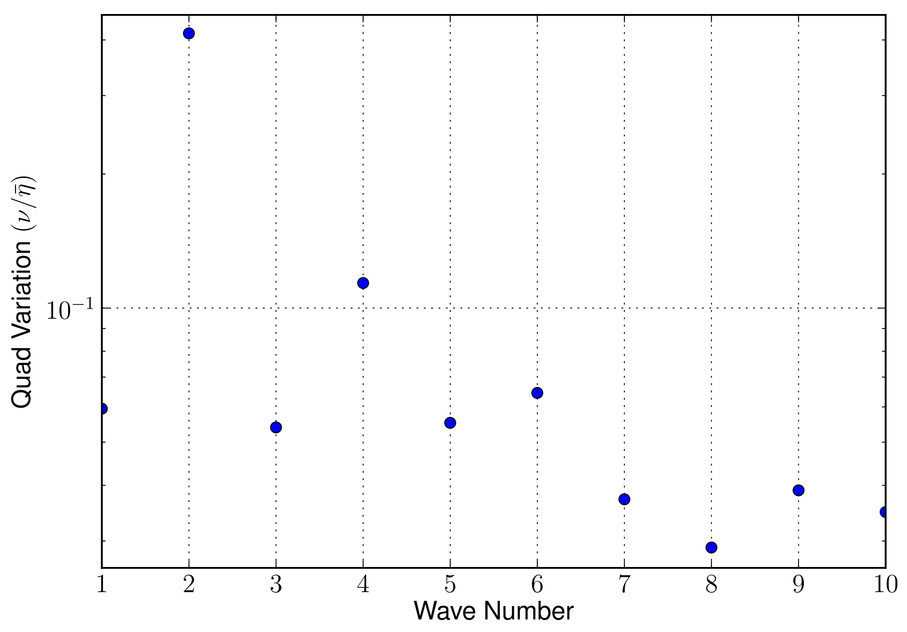


Figure 5.19: Harmonic analysis of the quadrature control signal. Quadrature control which is expected to be a good measure of the frequency mismatch and shows a large component at $\cos 2\theta$ and quickly decreasing components at $\cos 4\theta$ and $\cos 6\theta$.

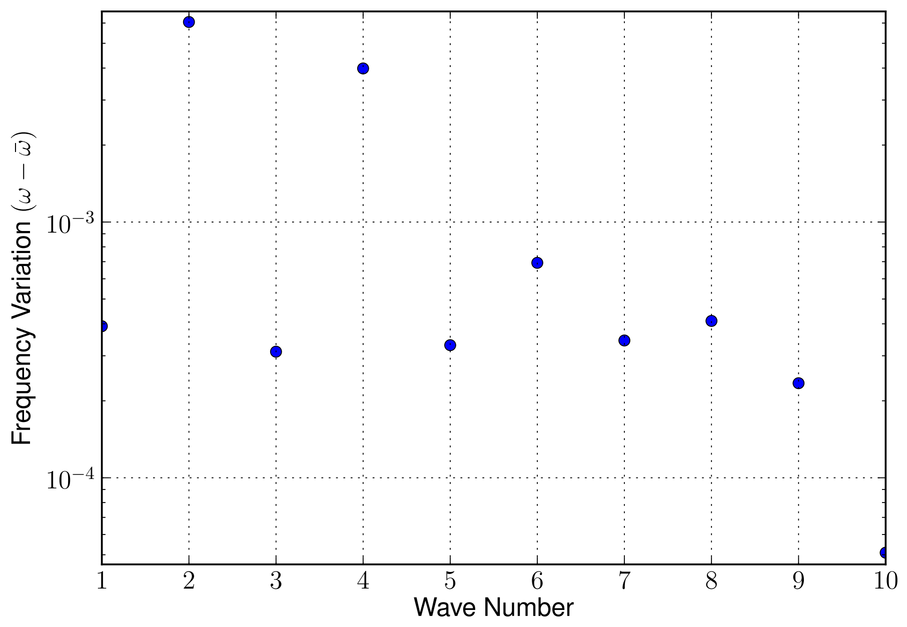


Figure 5.20: Harmonic analysis of the PLL frequency. The PLL frequency should match the resonant frequency for the current orientation. It has large components at $\cos 2\theta$ and $\cos 4\theta$ and a much smaller component at $\cos 6\theta$.

5.10 Hybrid Mode Rate Gyroscope Control

The rate control operates using rate-integrating control equations [19] with an orientation control enabled. The orientation control holds the gyroscope at a fixed orientation (ratio of X-axis and Y-axis amplitudes). The software parts of the control does not suffer from the limitations of the fixed-point control, but the delay in the USB bus limits the bandwidth. The delay is around 10 ms depending on the computer and the complexity of the controls, which limits the bandwidth to approximately 300 Hz in the best case. The controls could be ported to other digital signal processor (DSP) hardware with a faster link between FPGA and microprocessor to improve the bandwidth.

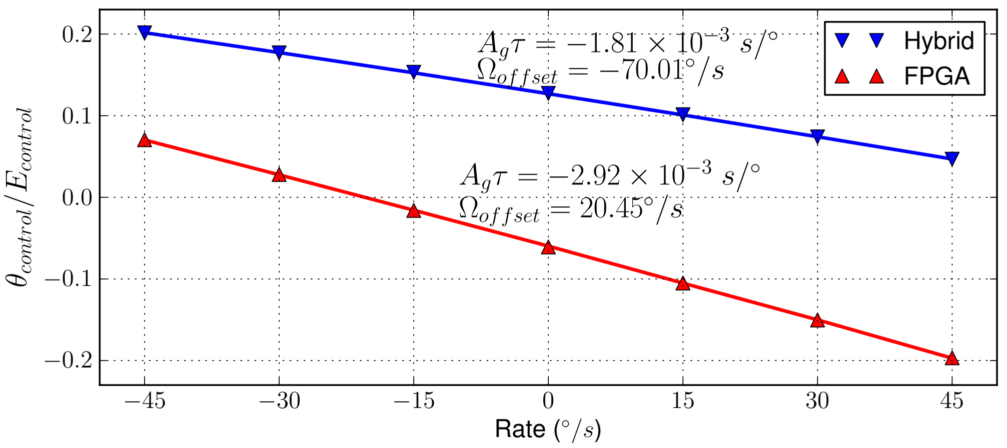


Figure 5.21: The sensitivity of a CING gyroscope with FPGA control and hybrid control. The sensitivity is defined as the ratio of the force-feedback to the amplitude control signal. The difference in offset and slope is due to different tuning, and for the hybrid control the orientation was held at 45°(X and Y equal) rather than 0°.

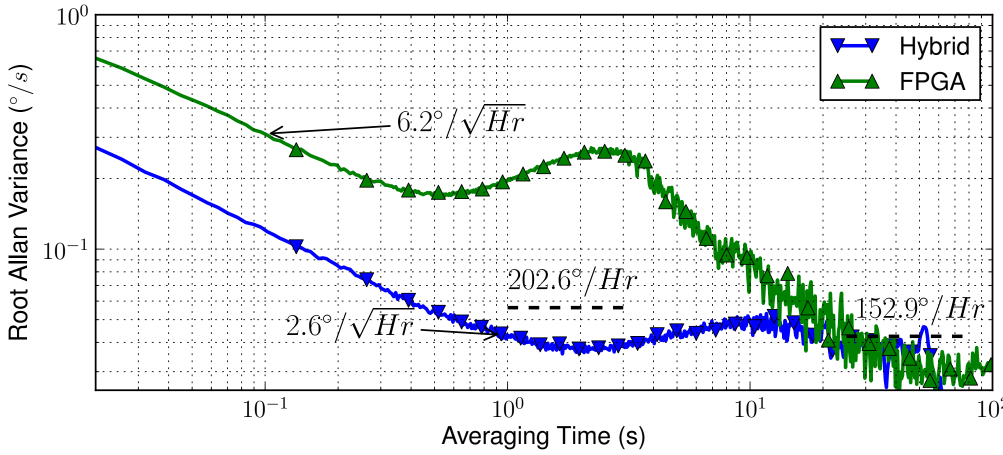


Figure 5.22: CING Rate Mode Allan Variance with FPGA and hybrid control. The bias stability is very sensitive to the control parameters, especially the integral gain which can induce small slow oscillations, visible here as the bumps at 10 second and 2 second averaging. Improvement in tuning and gain selection will improve performance.

Table 5.1: Performance of a CING gyro when controlled in rate mode with the FPGA rate control or the hybrid control.

	FPGA control	Hybrid control [35]
Angle Random Walk ($°/\sqrt{Hr}$)	6.2	2.6
Bias Stability (°/Hr)	153	203
A_g ($°/s^{-1}$)	0.02	0.011

5.11 Summary

The basic controls for a RIG have been implemented in software and hybrid software/firmware control systems. The delay in the software control makes stable operation difficult, while the hybrid mode has demonstrated stable operation of a MEMS RIG for hours at a time over the course of several days.

This control system has enabled the measurement of the gyroscope parameters as a function of the orientation and analysis of the harmonics of the error pattern reveal that there are large non-linearity which create components at $\cos 4\theta$ and $\cos 6\theta$ in addition to the standard models $\cos 2\theta$.

The drift in the test gyroscopes is very large due to the large mismatch, especially damping mismatch. New compensation methods to address this drift are presented in Chapter VI.

CHAPTER VI

Dynamic Mismatch Compensation

The basic controls of a RIG discussed in Chapter V work well for a macro-scale gyroscope where through careful fabrication and mechanical tuning the relative mismatch can be made extremely small. With a MEMS gyroscope however, especially as the resonator is made smaller, the relative mismatch will tend to increase and the capability to do mechanical tuning is decreased. Some methods of mechanical tuning of MEMS gyroscopes have been proposed [22, 57], this kind of tuning would significantly increase the cost of gyroscope and may not be sufficient to achieve the extremely high standard of matching required for a RIG. This chapter presents some advanced controls to improve MEMS RIG gyroscope performance. The error model in this chapter is based on the standard $\cos 2\theta$ mismatch distribution used in most of the literature. The effect of using this model with the non-linear gyroscope will be seen in Section 6.3 as well as improved performance from tuning the compensation values.

6.1 Novel Compensation Controls

The ideal MEMS RIG can be modeled as

$$\begin{aligned}\ddot{x} + \frac{1}{\tau}\dot{x} + 2A_g\Omega\dot{y} + \omega^2 x = 0 \\ \ddot{y} + \frac{1}{\tau}\dot{y} - 2A_g\Omega\dot{x} + \omega^2 y = 0\end{aligned} \tag{6.1}$$

where τ is the damping time constant, A_g is the mechanical angular gain, ω is the resonance frequency and Ω is the rate of rotation. When modeling the gyroscope, it can be useful to think of on-axis velocity terms as damping forces, cross-axis velocity terms as drift forces, and position terms as quadrature forces. Due to material properties and fabrication uncertainty, the damping and frequency will depend on the orientation of the oscillation. A commonly used model of the damping and frequency anisotropy is

$$\frac{1}{\tau} = \frac{1}{\tau_0} + \Delta\frac{1}{\tau}\cos 2(\theta - \theta_\tau) \tag{6.2}$$

$$\omega^2 = \omega_0^2 + \omega\Delta\omega\cos 2(\theta - \theta_\omega) \tag{6.3}$$

where θ_τ is the orientation where the damping is minimum, θ_ω the orientation where the resonance frequency is maximum, $\Delta\frac{1}{\tau} = \frac{1}{\tau_1} - \frac{1}{\tau_2}$ is the magnitude of the damping mismatch and $\omega\Delta\omega = \frac{\omega_1^2 - \omega_2^2}{2}$ the resonance frequency mismatch. The terms τ_1, τ_2, ω_1 and ω_2 are the damping and resonance frequency parameters at $\theta_{\tau,\omega}$ and $\theta_{\tau,\omega} + \pi/2$ respectively.

By combining (6.1) with (6.2) and (6.3), a model of the gyroscope which includes the mismatch errors can be derived [19]

$$
\begin{aligned}
&\ddot{x} - 2A_g\Omega\dot{y} + \frac{2}{\tau}\dot{x} + \Delta\left(\frac{1}{\tau}\right)(\dot{x}\cos 2\theta_\tau + \dot{y}\sin 2\theta_\tau) \\
&\quad + \omega^2 x - \omega\Delta\omega\,(x\cos 2\theta_\omega + y\sin 2\theta_\omega) = 0 \\
&\ddot{y} + 2A_g\Omega\dot{x} + \frac{2}{\tau}\dot{y} + \Delta\left(\frac{1}{\tau}\right)(-\dot{x}\sin 2\theta_\tau + \dot{y}\cos 2\theta_\tau) \\
&\quad + \omega^2 y - \omega\Delta\omega\,(-x\sin 2\theta_\omega + y\cos 2\theta_\omega) = 0.
\end{aligned}
\tag{6.4}
$$

6.1.1 Damping Mismatch Compensation Loop

In [58], a method is described for dynamically determining the magnitude and orientation of the damping mismatch in a RIG and adjusting the amplitude control to account for the variation in a gyroscope which has both a ring drive electrode and drive electrodes on the X and Y axes. The drift due to damping is not directly addressed and our own simulations show that although amplitude variation is removed, the drift is not. The method presented here does not use a ring electrode, which is not always available in MEMS RIGs, and directly addresses the drift due to damping mismatch. Also, the derivation presented here comes more intuitively from the gyroscope model than that presented in [58].

To maintain the gyroscope oscillation, it is necessary to drive the X and Y axes with a force in phase with the oscillation velocity. Examining (6.4) with only the damping mismatch terms and including forces to maintain steady-state behavior

$$
\begin{aligned}
\ddot{x} + \frac{1}{\tau}\dot{x} + \Delta\frac{1}{\tau}\dot{x}\cos 2\theta_\tau + \Delta\frac{1}{\tau}\dot{y}\sin 2\theta_\tau + \omega^2 x &= F_{ex} + F_{sx} \\
\ddot{y} + \frac{1}{\tau}\dot{y} - \Delta\frac{1}{\tau}\dot{y}\cos 2\theta_\tau + \Delta\frac{1}{\tau}\dot{x}\sin 2\theta_\tau + \omega^2 x &= F_{ey} + F_{sy}
\end{aligned}
\tag{6.5}
$$

where F_{ex} and F_{ey} maintains the oscillation amplitude and F_{sx} and F_{sy} compensates for any drift terms. By assigning the damping terms to F_{ex},F_{ey} and the drift terms to F_{sx},F_{sy}, the steady-state values of these forces to maintain the oscillation and compensate any drift are

$$\begin{aligned} F_{ex} &= \frac{1}{\tau}\dot{x} + \Delta\frac{1}{\tau}\dot{x}\cos 2\theta \\ F_{sx} &= \Delta\frac{1}{\tau}\dot{y}\sin 2\theta \\ F_{ey} &= \ddot{y} + \frac{1}{\tau}\dot{y} - \Delta\frac{1}{\tau}\dot{y}\cos 2\theta \\ F_{sy} &= \Delta\frac{1}{\tau}\dot{x}\sin 2\theta \end{aligned} \tag{6.6}$$

If the quadrature compensation loop maintains $a >> q$, then using (5.1) the velocities can be written as

$$\begin{aligned} \dot{x} &= \omega a \cos\theta \sin\omega t \\ \dot{y} &= \omega a \sin\theta \sin\omega t \end{aligned} \tag{6.7}$$

and by inserting (6.7) in to (6.6), the forces can be simplified to

$$\begin{aligned} F_{ex} &= (\eta\cos\theta + \eta_c\cos\theta)\sin\omega t \\ F_{ey} &= (\eta\sin\theta - \eta_c\sin\theta)\sin\omega t \\ F_{sx} &= (\eta_s\sin\theta)\sin\omega t \\ F_{sy} &= (\eta_s\cos\theta)\sin\omega t \end{aligned}$$

where the terms η, η_c and η_s are

$$\begin{aligned} \eta &= \omega a\frac{1}{\tau} \\ \eta_c &= \omega a\Delta\frac{1}{\tau}\cos 2\theta_\tau \\ \eta_s &= \omega a\Delta\frac{1}{\tau}\sin 2\theta_\tau \end{aligned}$$

The basic amplitude control will have a mean output of η. Returning to the analysis in [58], the terms η_c and η_s can be found by implementing controls on the amplitude modulated by $\cos 2\theta$ and $\sin 2\theta$ leading to the control equations

$$\begin{aligned}
\Delta E &= E - E_0 \\
\eta &= A_p \Delta E + \sum \left[A_i \Delta E + A_d (\Delta E - \Delta E_{-1}) \right] \\
\eta_c &= \sum \cos 2\theta \left[A_i \Delta E + A_d (\Delta E - \Delta E_{-1}) \right] \\
\eta_s &= \sum \sin 2\theta \left[A_i \Delta E + A_d (\Delta E - \Delta E_{-1}) \right]
\end{aligned} \tag{6.8}$$

where A_p, A_i, A_d are proportional, integral and derivative gains which can be chosen to optimize the response of the gyroscope, and the subscript $_{-1}$ indicates the previous value.

6.1.2 Frequency Mismatch Compensation Loop

A control loop to reduce the quadrature error is required for non-ideal RIGs. Since the rate of growth of the quadrature is angle dependent, the control loop will tend to over or under compensate at various orientations and depending on the rotation rate. The gyroscope equations with only the frequency mismatch non-ideal terms and forces to cancel the quadrature growth are

$$\begin{aligned}
&\ddot{x} + \omega^2 x - \omega \Delta\omega \left(x \cos 2\theta_\omega + y \sin 2\theta_\omega \right) = F_{qx} \\
&\ddot{y} + \omega^2 y + \omega \Delta\omega \left(-x \sin 2\theta_\omega + y \cos 2\theta_\omega \right) = F_{qy}
\end{aligned} \tag{6.9}$$

where F_{qx}, F_{qy} cancel growth in quadrature error from the frequency. There are other sources of quadrature error such as rotational acceleration and damping mismatch, however the frequency mismatch terms dominate for even small values of $\Delta\omega$. Since the form of (6.9) is similar to (6.5), it is proposed that a similar control scheme can be used to improve the response to resonance frequency variation with θ.

Setting the cancellation forces equal to the frequency mismatch terms gives

$$F_{qx} = -\omega\Delta\omega\,(x\cos 2\theta_\omega + y\sin 2\theta_\omega)$$
$$F_{qy} = +\omega\Delta\omega\,(-x\sin 2\theta_\omega + y\cos 2\theta_\omega)$$

which, using the relationship (6.7) and assuming $a >> q$, can be rewritten as

$$F_{qx} = -\left(\nu_c\sin\theta + \nu_s\cos\theta\right)\cos\omega t$$
$$F_{qy} = +\left(\nu_c\cos\theta - \nu_s\sin\theta\right)\cos\omega t$$

where the terms ν_c and ν_s are

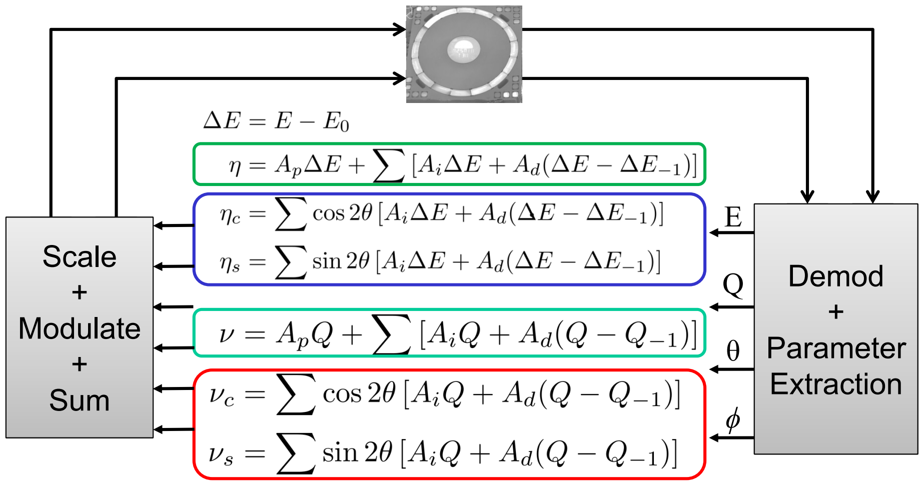
$$\nu_c = \omega\Delta\omega a\cos 2\theta_\omega$$
$$\nu_s = \omega\Delta\omega a\sin 2\theta_\omega$$

By an analysis identical to that for damping mismatch compensation, the following control laws are proposed

$$\begin{aligned}\nu &= A_pQ + \sum\left[A_iQ + A_d(Q - Q_{-1})\right]\\ \nu_c &= \sum\cos 2\theta\left[A_iQ + A_d(Q - Q_{-1})\right]\\ \nu_s &= \sum\sin 2\theta\left[A_iQ + A_d(Q - Q_{-1})\right]\end{aligned}\tag{6.10}$$

where ν_c and ν_s will compensate the quadrature growth due frequency mismatch and ν will control the quadrature growth from other sources. The final quadrature cancellation signals including ν are then

$$F_{qx} = -\left((\nu+\nu_c)\sin\theta + \nu_s\cos\theta\right)\cos\omega t$$
$$F_{qy} = +\left((\nu+\nu_c)\cos\theta - \nu_s\sin\theta\right)\cos\omega t.$$



$\Delta E = E - E_0$
$\eta = A_p \Delta E + \sum [A_i \Delta E + A_d(\Delta E - \Delta E_{-1})]$
$\eta_c = \sum \cos 2\theta [A_i \Delta E + A_d(\Delta E - \Delta E_{-1})]$
$\eta_s = \sum \sin 2\theta [A_i \Delta E + A_d(\Delta E - \Delta E_{-1})]$
$\nu = A_p Q + \sum [A_i Q + A_d(Q - Q_{-1})]$
$\nu_c = \sum \cos 2\theta [A_i Q + A_d(Q - Q_{-1})]$
$\nu_s = \sum \sin 2\theta [A_i Q + A_d(Q - Q_{-1})]$
Scale + Modulate + Sum
Demod + Parameter Extraction
E
Q
θ
ϕ

Figure 6.1: Diagram of dynamic mismatch compensation RIG control. The dynamic mismatch loops complement the normal control loops and can be enabled and disabled without disrupting the gyroscope operation.

The feedback loops used to implement dynamic mismatch compensation are illustrated in Figure 6.1.

If the frequency mismatch is large, the required F_{qx} and F_{qy} may be larger than can be supported by the control hardware. Also, large quadrature cancellation signals are undesirable since they can saturate the sense circuits due to capacitive feed-through or cause error in the amplitude detection if not perfectly removed by feed-through cancellation. To reduce the level of the quadrature drive signals, ν_c and ν_s can be used to adjust DC bias voltages which reduce the mismatch terms by electrostatic spring softening. ν_c and ν_s can be transformed into effective stiffness mismatch and cross coupling parameters by

$$\Delta k = s_k \omega_0 2\pi \frac{\sqrt{\nu_c^2 + \nu_s^2}}{\eta \tau}$$

$$s_k = \begin{cases} -1 & \text{if } \nu_s > 0 \\ 1 & \text{if } \nu_s \leq 0 \end{cases}$$

$$k_{xy} = -2\Delta k \arctan \frac{\nu_c}{\nu_s}.$$

An automatic mode matching routine such as that described in Section 3.2 can then be used to determine the bias voltages, keeping the mismatch within a range that can be compensated for by the dynamic technique above.

6.2 Simulation Results

The frequency and amplitude control signal are plotted versus orientation for a gyroscope with only frequency mismatch in Figure 6.2. With only quadrature control enabled, there is no variation of the damping signal against orientation and the resonant frequency depends on the orientation. The resonance frequency is provided by the all-digital phase-locked loop (ADPLL) in the simulation and with a physical gyroscope. With the frequency mismatch control enabled, the frequency is forced to a constant value independent of the orientation and an apparent damping mismatch appears. With both frequency and damping compensation enabled, both the frequency and amplitude control become independent of the orientation.

The drift for the same gyroscope is plotted versus orientation in Figure 6.4. The drift with quadrature control and frequency compensation is nearly identical. The small shift is probably due to not allowing the control to fully settle. By enabling both the frequency and damping compensation, the residual drift is reduced to nearly zero.

In a gyroscope with both damping and frequency mismatch, there is a dependency on orientation for both the amplitude control and frequency as plotted in Figure 6.3. Enabling the frequency compensation increases the damping mismatch somewhat, while enabling the damping compensation does not significantly affect the frequency mismatch. Enabling both eliminates the dependency on orientation for both amplitude control and frequency. The drift for the different controls is plotted in Figure 6.5.

The drift for the same gyroscopes as Figure 5.10 but with the dynamic mismatch

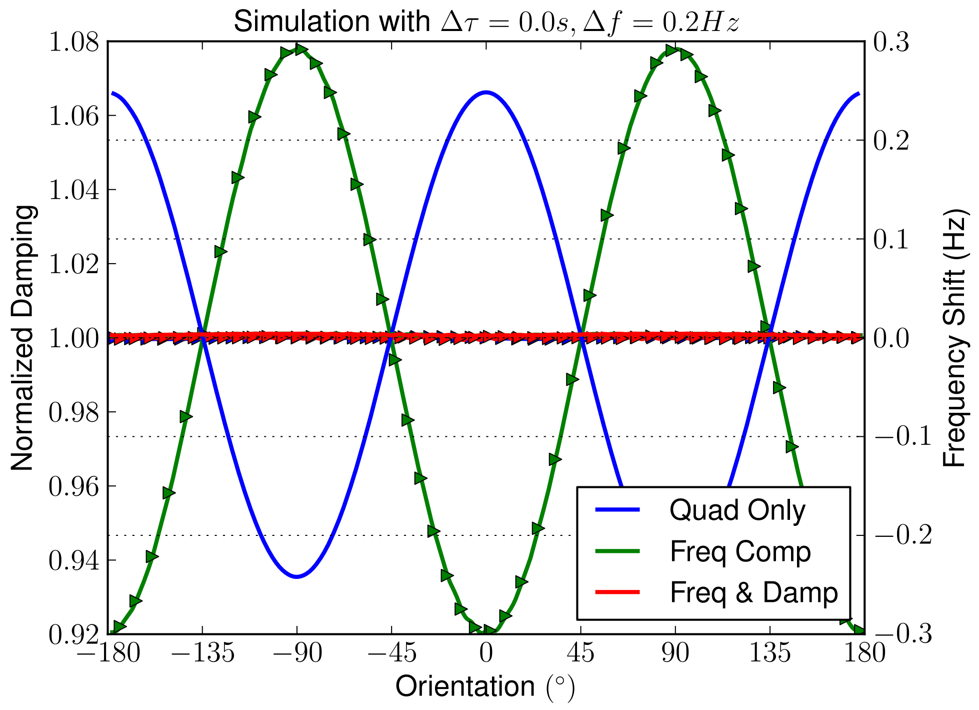


Figure 6.2: Simulated frequency and damping control with dynamic compensation and no damping mismatch.

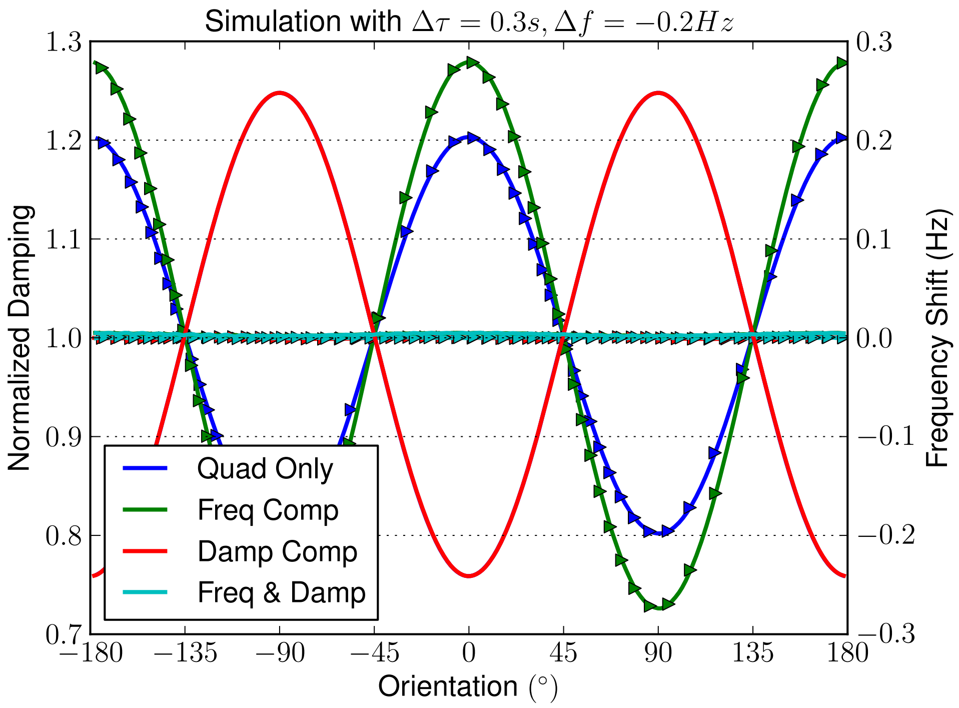


Figure 6.3: Frequency and damping control with dynamic compensation and both frequency and damping mismatch.

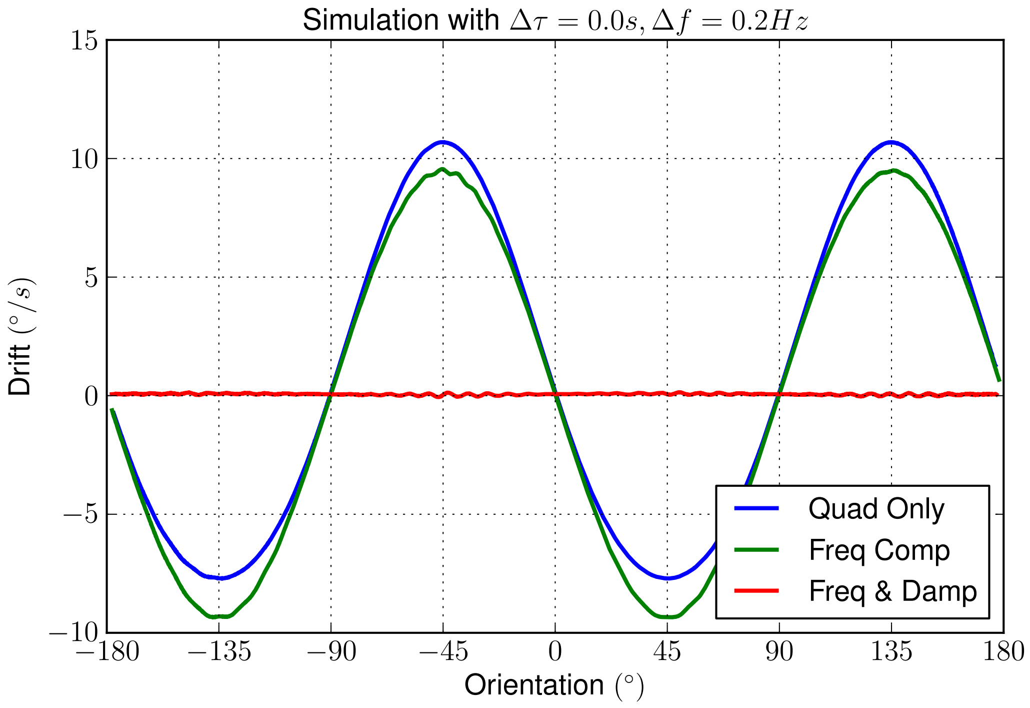


Figure 6.4: Drift of a gyro with only frequency mismatch.

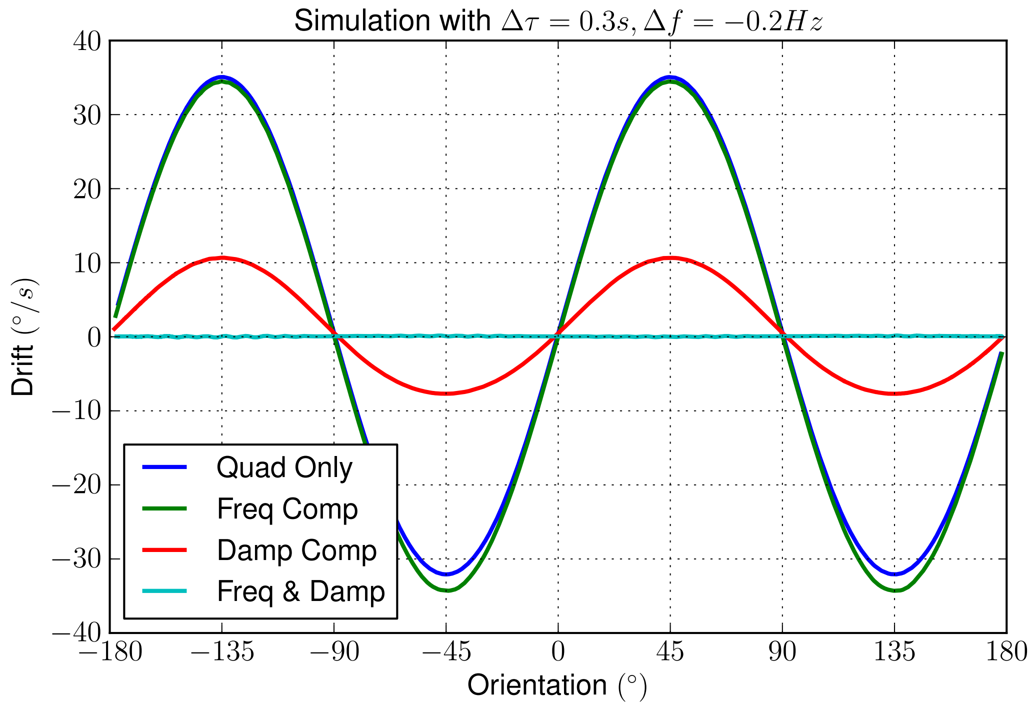


Figure 6.5: Drift of a gyro with frequency and damping mismatch.

control enabled is plotted in Figure 6.6. The frequency mismatch control loop settles within 9 seconds with non-optimized control values when a $90°/s$ rotating drive signal is applied, however the damping mismatch takes much longer to settle, over 266 seconds as shown in Figure 6.7. The damping mismatch takes a long time to settle because the deviations of E from the set value with the amplitude control enabled are very small. An alternate method of generating the control values based on the derivative of the amplitude control

$$\begin{aligned}
\Delta E &= E - E_0 \\
\eta &= A_p \Delta E + \sum \left[A_i \Delta E + A_d (\Delta E - \Delta E_{-1}) \right] \\
\eta_c &= \sin 2\theta \sum \left[A_i \Delta E + A_d (\Delta E - \Delta E_{-1}) \right] \\
\eta_s &= -\cos 2\theta \sum \left[A_i \Delta E + A_d (\Delta E - \Delta E_{-1}) \right]
\end{aligned} \tag{6.11}$$

settles much faster (under 14 seconds with only $60°/s$ applied rate), as in Figure 6.8

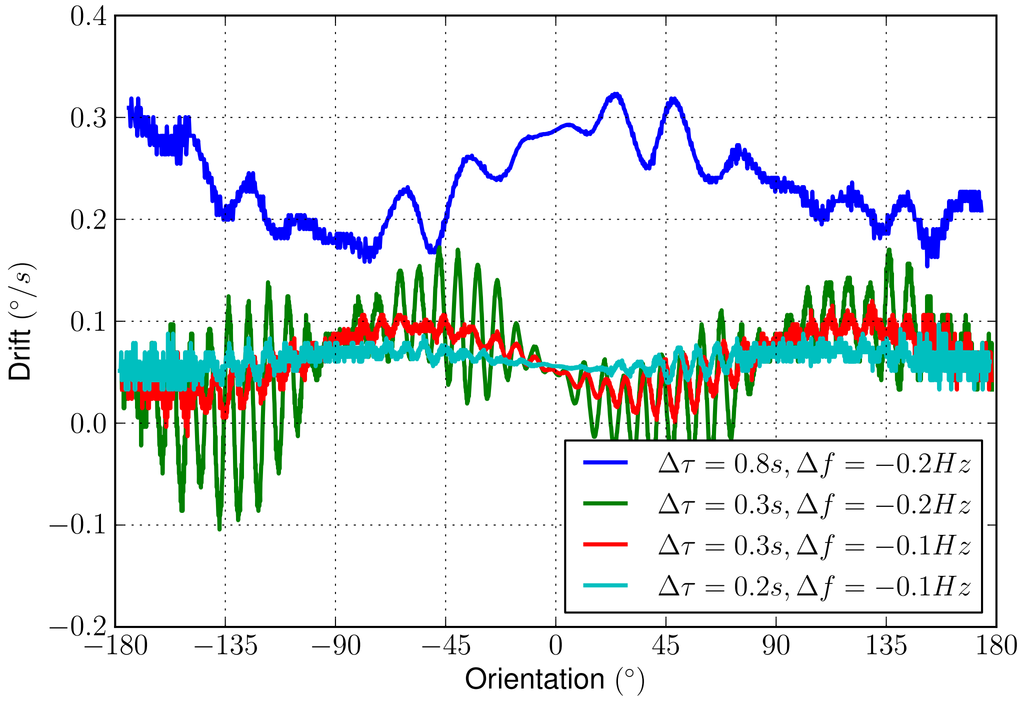


Figure 6.6: Simulated drift versus orientation with damping and frequency mismatch control. The simulated gyroscope has a nominal frequency of 5 kHz and Q of 25,000 ($\tau = 1.6sec$) with various amounts of mismatch with both the proposed damping and mismatch compensation enabled. The ripples in the remaining drift are due to the settling of the PLL after enabling the compensation controls.

but this method is unstable at low rates of rotation. One solution would be to have the control automatically switch between the two methods when the rotation rate crosses some threshold.

In the above plots, simulated rates of rotation were applied to the gyroscopes in order to move the wave orientation. It is also possible to use electric signals to move the wave orientation or create a simulated rotation. However, the lack of centripetal force and angular acceleration mean the electronic rotation is not the same as physical rotation. Also, in physical systems, various out of plane vibrations can be coupled into the in-plane vibrations by rotation which is not the case with electrical rotation.

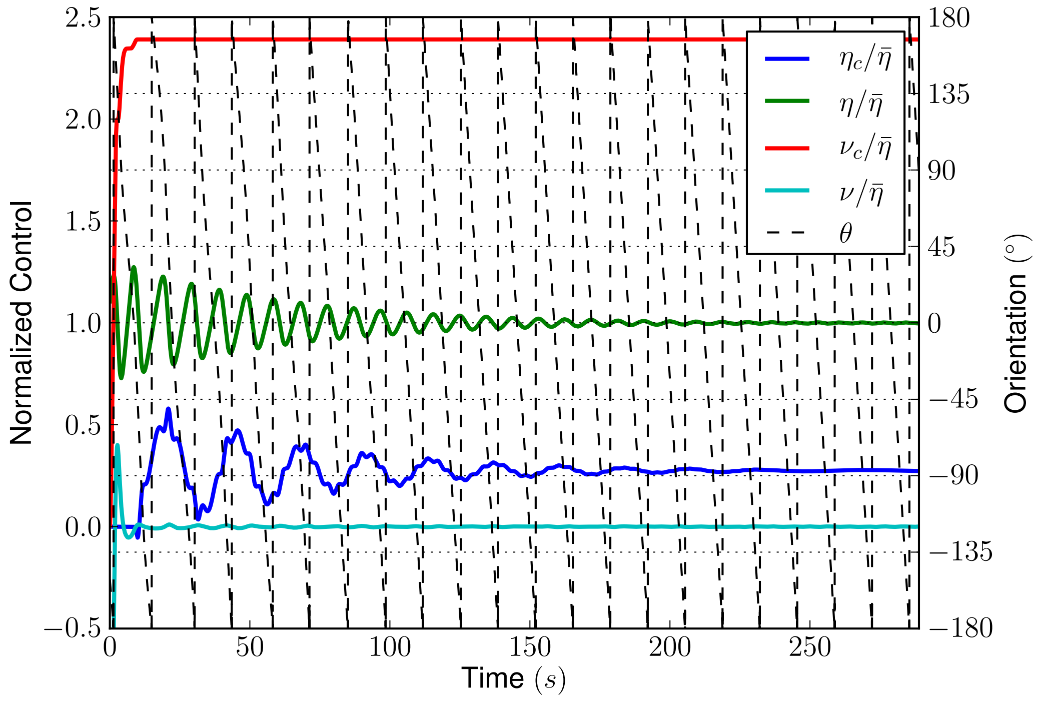


Figure 6.7: Simulated mismatch control settling. In this simulation, the frequency mismatch loop, ν_c settles to 1% in 9 seconds and the damping mismatch loop, η_c, in 266 seconds with $90^\circ/s$ applied rate of rotation.

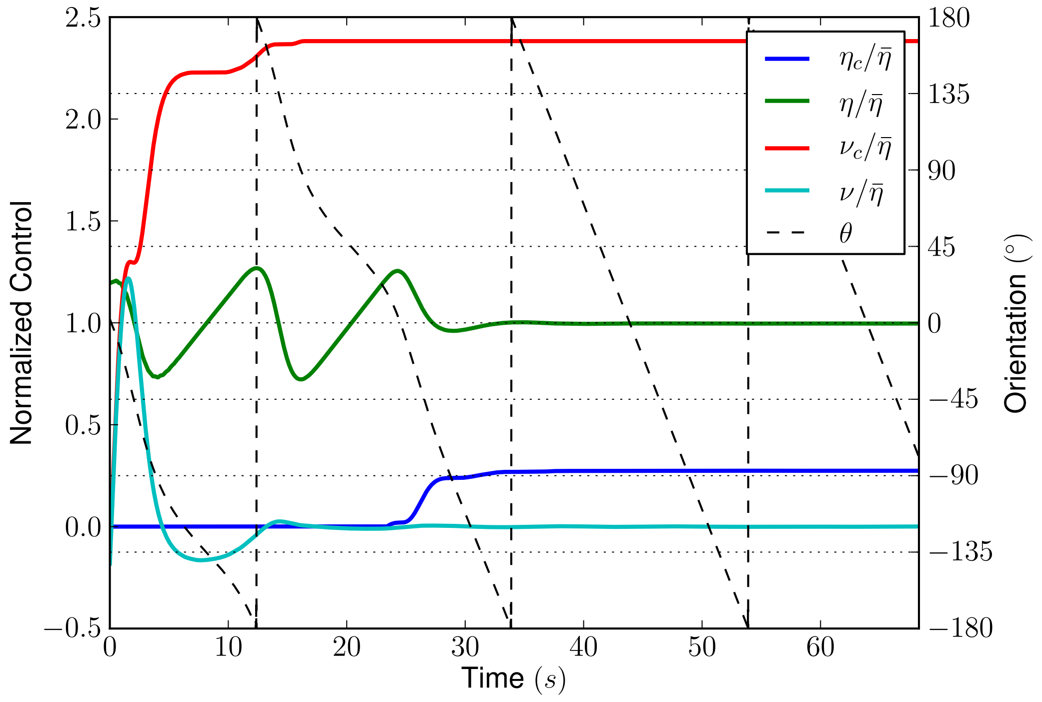


Figure 6.8: Simulated mismatch control settling with modified control law. With the modified control, (6.11), where the damping mismatch is extracted from the derivative of the damping variation, settling of the damping control is much faster. Here, ν_c settles to 1% in 13.5 seconds and the damping mismatch loop, η_c, in 13.8 seconds with $60°/s$ applied rate of rotation.

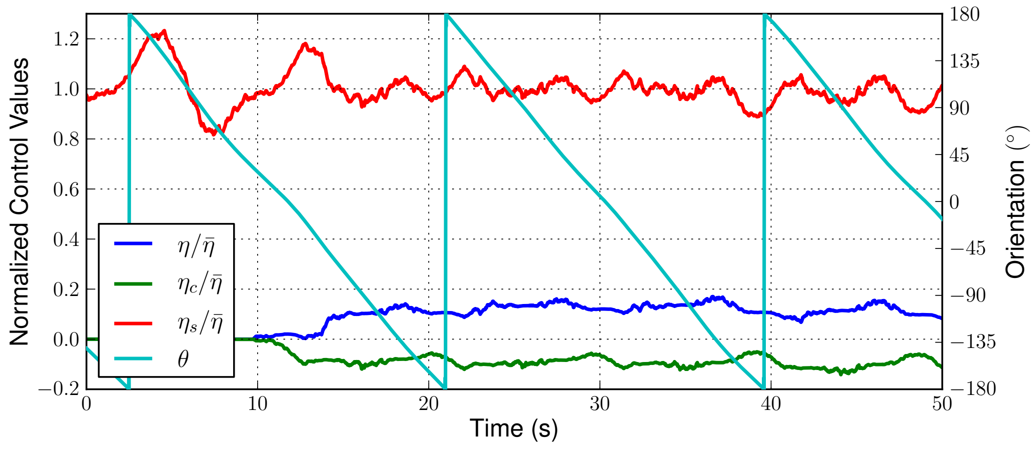


Figure 6.9: Damping mismatch control with a CING gyroscope. Enabling the damping mismatch control with CING gyroscope reduces the amplitude control variation and removes some of the angle dependency of the rate scale factor. The control is enabled at 10 seconds and has mostly settled within 3 seconds.

6.3 Measurement Results

The damping mismatch control loop is shown to settle in 3 seconds when driven with an rotating drive signal in Figure 6.9. The frequency mismatch control loop settles in 7 seconds in Figure 6.10. The tests were made without optimizing the control gains. The inability of the control loops to fully settle is due to the measured errors not matching the model in (6.2) and (6.3) as discussed in Section 5.9.

Despite the mismatch between the model and the gyroscope errors, the controls are still able to reduce the drift of a CING gyroscope as shown in Figure 6.11. The RMS drift is reduced by 25%, and the remaining drift is dominated by the regions where the model and measured error are the most different. The source of this difference could be particular to the CING, the hardware implementation, or a more general problem with MEMS RIG. The error model can also be updated to better approximate the measured error and further improve performance.

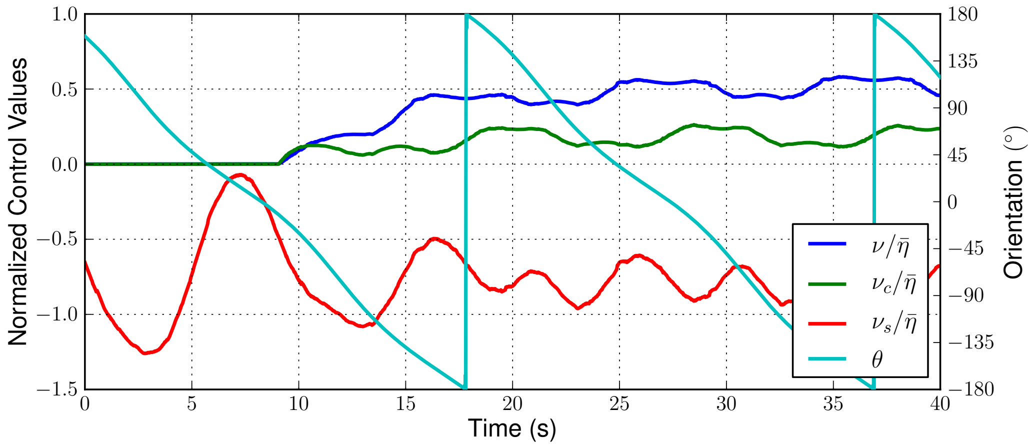


Figure 6.10: Frequency mismatch control with a CING gyroscope. Enabling the frequency mismatch control with CING gyroscope reduces the quadrature control variation and the residual quadrature signature.The control is enabled at 9 seconds and is mostly settled within 7 seconds.

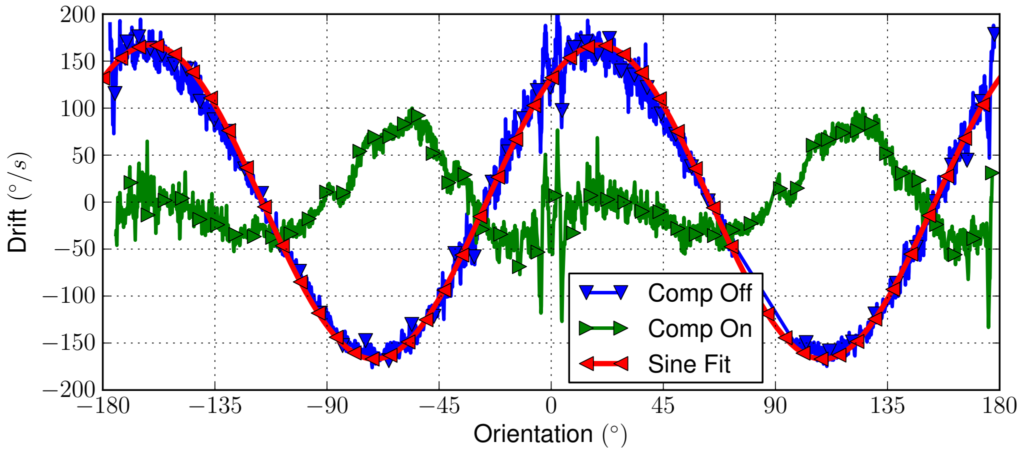


Figure 6.11: Drift with damping and mismatch enabled. The drift as measured from the variation of the measured rotation rate from the average rotation rate of a CING gyroscope under electronic rotation with and without the proposed compensation enabled. With this level of mismatch, no rate less than nearly $200°/s$ would fully rotate the gyroscope orientation.

6.3.1 Tuned Mismatch Compensation

Using the steering or angle control to cause the gyroscope orientation to process distorts the results of the dynamic mismatch compensation loop. This is likely due to coupling between the horizontal and vertical components of the mode as it precesses. Another potential reason is feed-through, but the feed-through is small and feed-through compensation does not significantly affect the result. In Figure 6.12, the mean output of the dynamic damping mismatch controls are plotted against the steering signal. The zero crossing is expected to be the correct value for compensation with no steering signal applied. The dynamic mismatch compensation can be applied using fixed values instead of activating the feedback loops. The drift with the dynamic mismatch compensation using the zero crossing values for the compensation, tuned DMC, is shown in Figure 6.13. The RMS drift is reduced by over 85% and the max drift is reduced by over 88%. The worst drift is reduced from -322°$/s$ to -41°$/s$. There is no uncompensated plot with applied rates because the rate table could not apply a large enough rate to process this gyroscope without the compensation. Using these fixed compensation values, the higher order terms of the damping variations are also reduced as illustrated in Figure 6.14. It is not clear why the mismatch compensation would reduce the higher order terms of the angle dependency of the damping.

To verify the angle measuring nature of these controls, a CING gyroscope with tuned dynamic mismatch compensation applied was subjected to alternating clockwise and counter-clockwise step rotations. The results for ±90° , ±180° , and ±270° rotations are plotted in Figures 6.15, 6.16, and 6.17 respectively. The angular gain is calculated for each step from the measured angle change divided by applied rotation and the average is 0.0115 which agrees very closely with the angular gain calculated from the constant rate tests in Section 5.9. This is the third method used to calculate the angular gain, in addition to the rate-mode slope and the time required for the standing wave to precesses fully, and the results from all tests agree closely. The

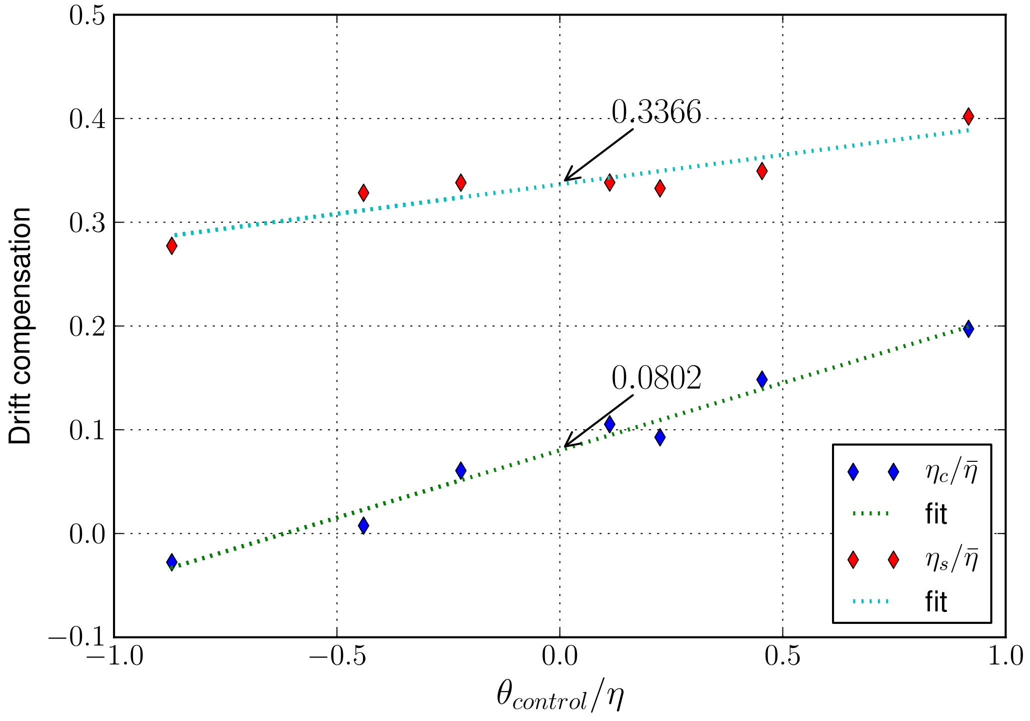


Figure 6.12: Damping mismatch control output versus steering signal. The steering control distorts the damping mismatch, possibly due to coupling of the vertical and horizontal components of the CING resonant mode. The zero crossing is expected to reflect the true mismatch.

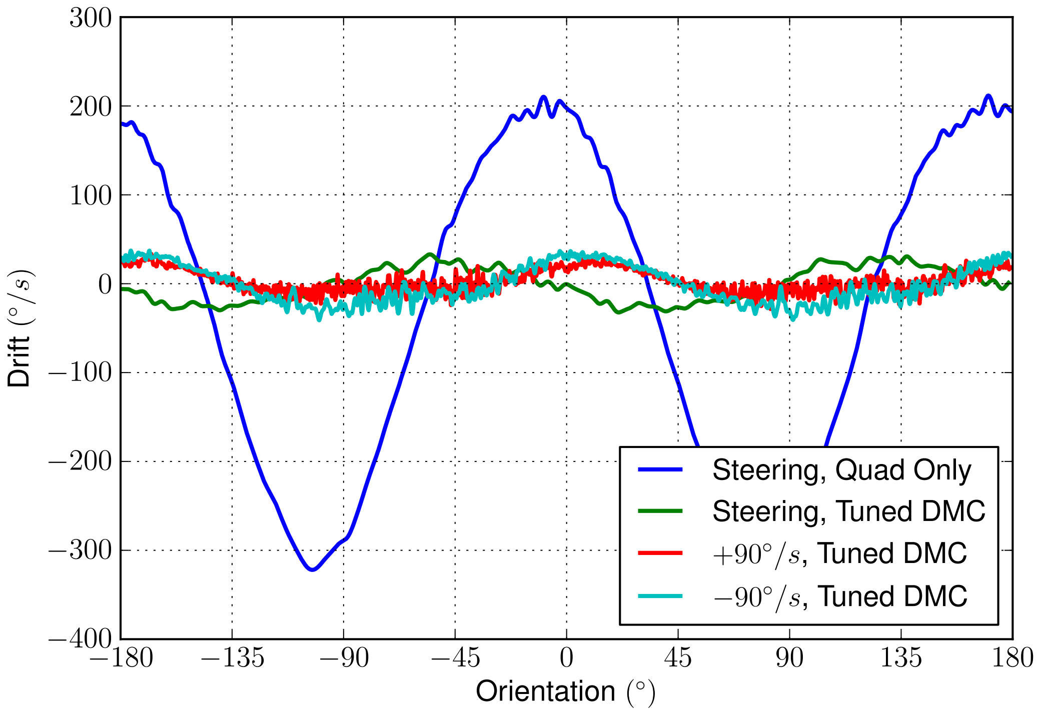


Figure 6.13: Drift of CING gyroscope with tuned damping and frequency mismatch compensation. Without compensation, the gyroscope will not fully precess with the rates that can be applied using the rate table (upto $120°/s$). With the tuned mismatch compensation (tuned DMC), precession is possible with both $\theta_{control}$ and applied rates of $\pm 90°/s$.

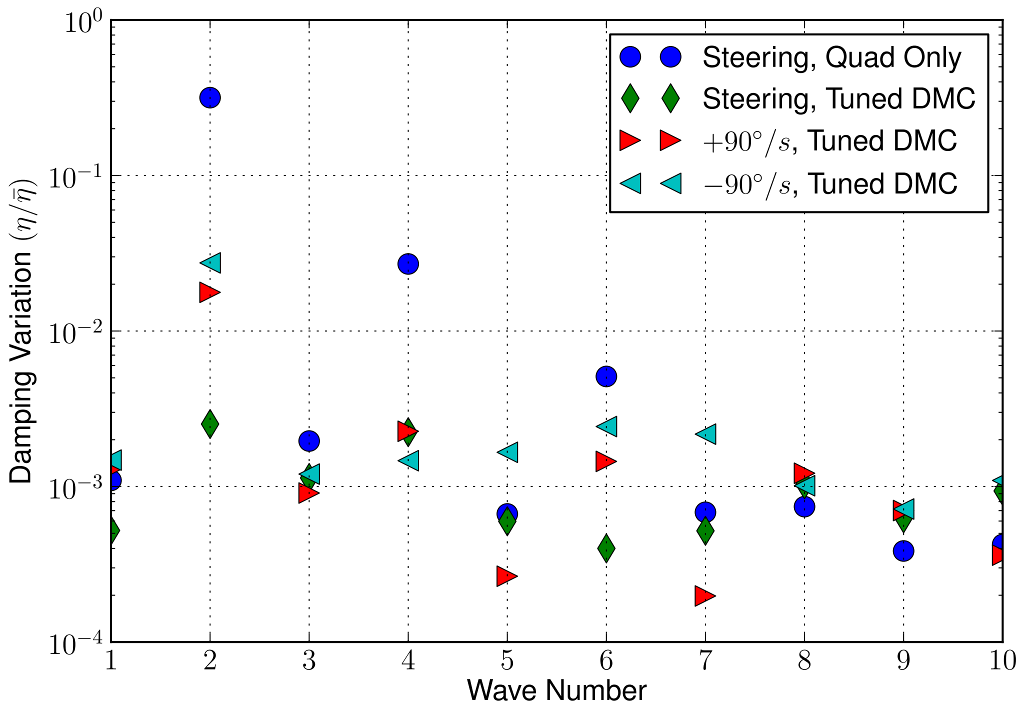


Figure 6.14: Damping harmonics with tuned damping and frequency mismatch compensation. The damping variation is measured using the amplitude control variation. All of the even terms are reduced with the mismatch compensation, independent of whether to oscillation is precessed with a real rate or using the steering or angle control.

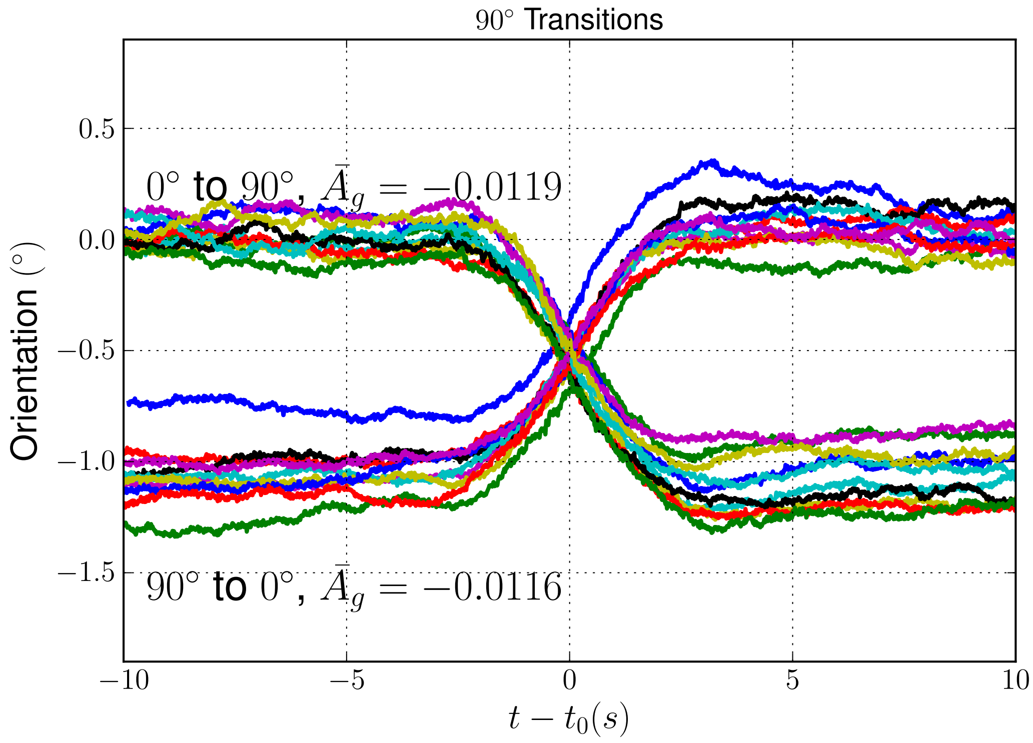


Figure 6.15: Oscillation angle change under ±90° step rotations.

similar behavior in these different tests indicate that the underlying model of the operation is correct.

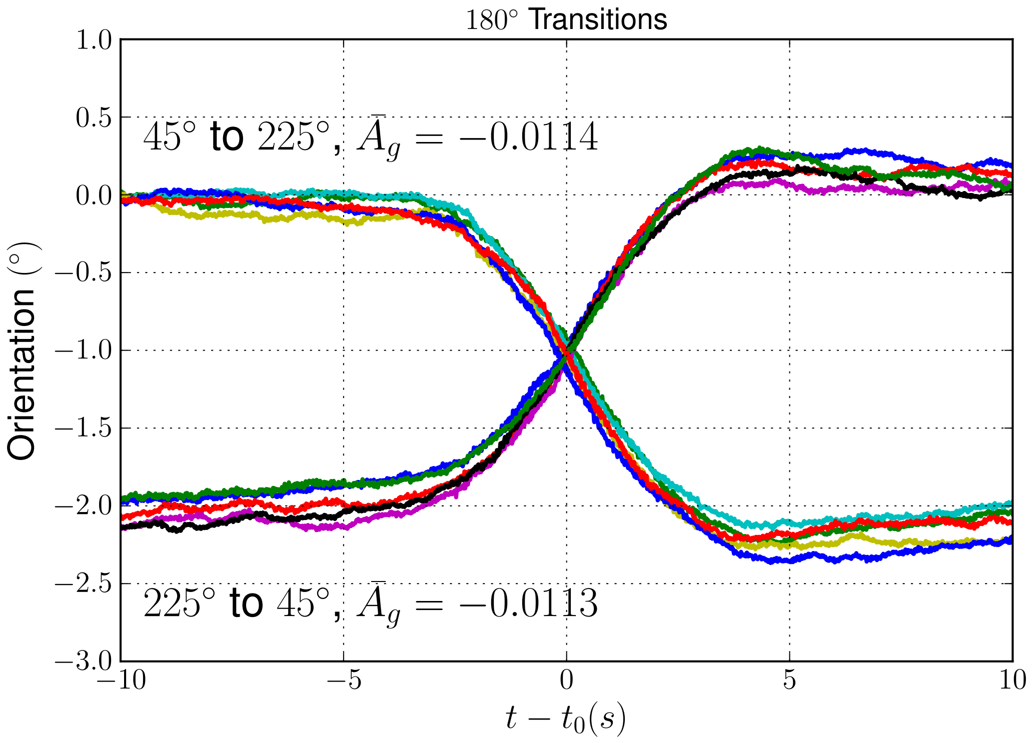


Figure 6.16: Oscillation angle change under ±180° step rotations. The rotation is between 45° and 225° instead of starting from 0°.

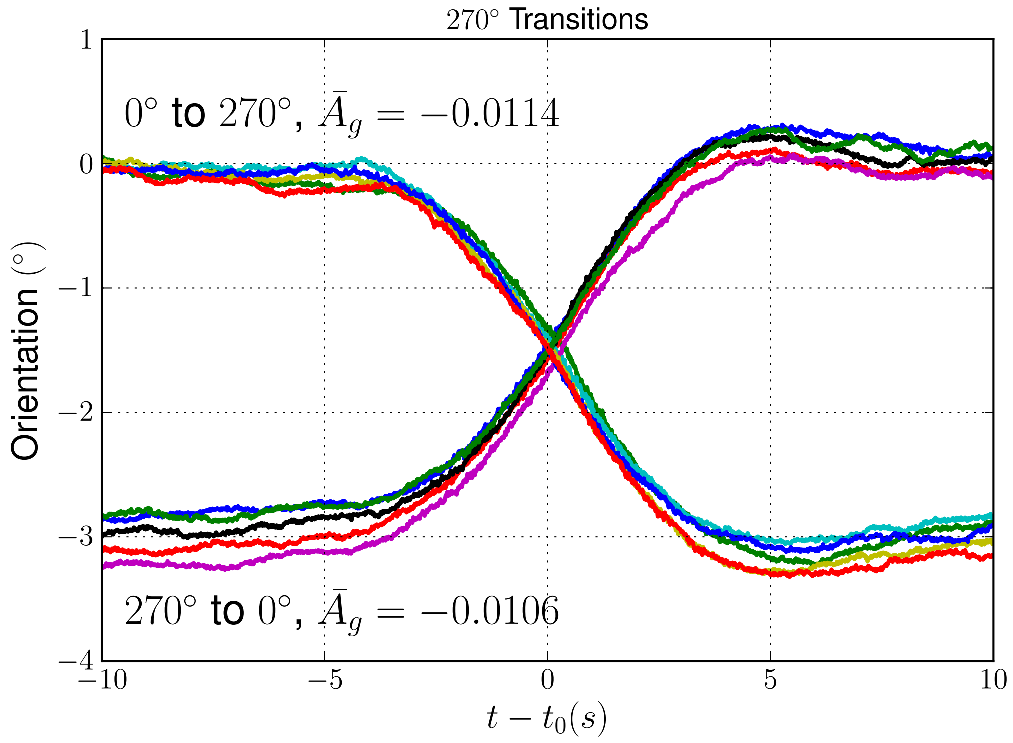


Figure 6.17: Oscillation angle change under ±270° step rotations.

6.4 Summary

We have described two novel control loops to characterize and compensate for damping and frequency mismatch in RIG, which we expect to be critical for high-performance operation of MEMS RIG. The loops use the residual error in the traditional control loops to determine orientation dependent error and an orientation steering control then cancels the drift induced by the mismatch. The controls have been tested in simulation and implemented in a functional control system and tested with a CING gyroscope. The deviation of the damping and quadrature behavior from the model limits the ability of the feedback loops to determine the correct compensation parameters, however using tuned parameters the drift can be reduced by over 85% from a worst case value of over $320°/s$to around $41°/s$.

CHAPTER VII

Conclusion and Contributions

We have presented control and compensation systems for vibratory MEMS gyroscopes to characterize high-Q (Q over 50,000) resonators and operate them as rate or rate-integrating gyroscopes. Although this work focused on gyroscopes, the same tools and controls can be used with accelerometers. MEMS Rate and Rate-Integrating gyroscopes fill key technical needs for consumer, military and space applications. MEMS gyroscopes offer advantages not only in size and weight, but in some harsh environments they may even outperform their meso-scale counterparts. While MEMS rate gyroscopes are well established, the RIG is an emerging component of the MEMS gyroscope field. The proposed research addresses several problems to be solved in MEMS RIG research. The problems to be addressed are slow characterization and tuning, poor performance due to damping mismatch, the lack of robust and accessible control hardware and algorithms, and the difficulty in using RIG due to the overall complexity of tuning, tweaking, readout and control.

7.1 Contributions

- A characterization and control system for MEMS gyroscopes built on open and commercially available hardware and software. The characterization tools are much more complete than other work and include ring-down analysis, impulse response, gain-phase analysis and automatic tuning. The system is easy to use and expand so that other researchers can test their own devices or algorithms without large investment in hardware or time.

- Automatic mode-matching of high-Q (Q over 50,000) MEMS gyroscopes from over 40 Hz mismatch to below 100 mHz.
- An efficient FPGA implementation of rate gyroscope control with amplitude, rate and quadrature closed-loop control demonstrating better than 400% improvement in performance for a MEMS gyroscope over open-loop control.
- Hybrid software and firmware control system and gyroscope simulator for development of advanced gyroscope control algorithms and experimental testing of those algorithms. The hybrid architecture allows delay sensitive components to be implemented in the FPGA while other controls can be implemented in software where development is easier.
- Rate-integrating MEMS gyroscope operation for much much longer than the ring-down time as well as and characterization of the anisotropy, which is critical to development of improved models.
- New control algorithms for rate-integrating gyroscopes to dynamically measure damping and frequency anisotropy and compensate angle drift. Compensating angle drift allows rate-integrating operation at lower rates so that dual-mode MEMS gyroscopes are practical.

7.2 Future Work

The system described in this thesis is meant to be a test-bed for development of algorithms for MEMS gyroscopes and characterization of new resonators. To prove the applicability of this system to a wide range of gyroscopes, it should be tested with more gyroscopes than just the CING. Also, the controls have demonstrated basic operation of the CING gyroscope in a rate-integrating mode, but the inherent limitations of the CING make it difficult to evaluate more subtle problems with the control. Testing the system with other gyroscopes will help to identify problems caused by the control system.

Although the dynamic mismatch compensation shows promise for enabling medium quality gyroscopes to operate as RIG, more analysis is required to prove the stability of the controls under all situations, to determine the limits of where the controls are actually able to improve performance, and improve the error model to separate electrical and mechanical non-linearity and further reduce drift.

The time-multiplexing technique briefly introduced for control of multiple rate-gyroscopes could also used to control a signal gyroscope without introducing problems due to feed-through. This could significantly improve the ability to characterize and compensate RIG and can be extended for RIG operation.

Finally, transferring all of these techniques to a fully embedded system would greatly reduce system size and improve performance, especially due to reduced delay. A self-contained but still programmable version of the current system would allow rapid testing of algorithms for very high performance gyroscopes and rapid characterization of gyroscope resonators during the fabrication process.

APPENDICES

The following sections list the source code that is original to this work as is important to the measured and simulation results presented in this work.

APPENDIX A

Gyroscope Simulation Code

```
#include <math.h>
#include <complex>
#include <stdio.h>
#include <string.h>
#include <gsl/gsl_errno.h>
#include <gsl/gsl_matrix.h>
#include <gsl/gsl_odeiv.h>
#include <vector>
#include <string>
#include <gr_io_signature.h>
#include <gr_sincos.h>
#include <gyro_relay_model_source.h>
#include "gyro_common.h"
#include "fpga_regs_common.h"
#include "fpga_regs_standard.h"

#define SENSE_ROTATED  // read is rotated from drive
//#define POSITION // use position readout
double fixed2float16(int fix)
{
  return (fix >= TWO15 ? fix - TWO16 : fix) / TWO16;
}

double fixed2float32(int fix)
{
  return (fix >= TWO31 ? fix - TWO32 : fix) / TWO32;
}

gyro_relay_model_source::gyro_relay_model_source
(gr_msg_queue_sptr msgq,
 std::string fpga_model, double signal_rate, int upsample, double x0,
 double y0, double Th0, double ThS, const std::vector<double> &feedthrough,
 const std::vector<double> &gains, const std::vector<double> &tuning,
 double rate, double rate_freq, double wx, double wy, double th_frq,
 double txx, double tyy, double th_tau, double kr, double kc)
  : gr_sync_block ("relay_model_source",
                   gr_make_io_signature (0, 0, 0),
                   gr_make_io_signature (1, 1, sizeof (float))),
    d_msgq(msgq), d_signal_rate(signal_rate), d_upsample(upsample),
    d_rate(rate), d_rotated(0), d_rate_freq(rate_freq), d_rate_accel(1e6),
    d_kr(kr), d_kc(kc), d_tolerance(1e-9), d_iptr(NULL),
    d_vs((wx+wy)/2.), d_dt(1/signal_rate), d_irate(0.),
    d_drate(0.), d_rate_phase(0.), d_pp(0.0), d_pv(0.0), d_ld1(0.), d_ld2(0.),
    d_loopback(false), d_rxa(true), d_rxb(false), d_nch(2), d_rx_swap(false),
    d_tx_swap(false), d_rxenable(false), d_txenable(false),
    d_tuning_update(false),
    d_txphase(0), d_rxphase(0), d_pll_filt(gains, gains) /*will overwrite*/,
    d_pll_pid(327670, 0, 1), d_pll_updated(false), d_pll_f0(0), d_pll_delay(0),
    d_count(0)
{
  d_w0  = (wx*wx+wy*wy)/2.;
  d_wdw = (wx*wx-wy*wy)/2.;
  d_dmp = (1/txx+1/tyy);
  d_ddp = (1/txx-1/tyy);
  d_rwc = cos(2*th_frq);
  d_rws = sin(2*th_frq);
  d_rtc = cos(2*th_tau);
  d_rts = sin(2*th_tau);
  d_SthS= sin(2*ThS);
  d_CthS= cos(2*ThS);
  for(int it=0; it<4; it++)
    {
      d_ftg.push_back(feedthrough.at(it));
      d_ftc.push_back(feedthrough.at(it));
      d_gain.push_back(gains.at(it));
      d_Ks.push_back(tuning.at(it));
    }
  for(int it=0; it<2; it++)
    {
      d_new_pll_a.push_back(0);
      d_new_pll_b.push_back(0);
    }
  d_gain_range.push_back(-20);
  d_gain_range.push_back( 20);
  d_gain_range.push_back(0.5);
  d_step_type = gsl_odeiv_step_rkf45; //bsimp; //rk8pd;
  d_step      = gsl_odeiv_step_alloc(d_step_type, ODE_DIM);
  d_control   = gsl_odeiv_control_y_new(d_tolerance,
                                        d_tolerance*1e3);
  d_evolve    = gsl_odeiv_evolve_alloc(ODE_DIM);
  // jac is only used for some solvers,
  // it is complicated due to control signals and rate
  d_sys       = {relay_func, NULL /*jac*/, ODE_DIM, this};
  d_state[0]  = x0; d_state[1] = 0.; d_state[2] = y0; d_state[3] = 0.;
  if(fpga_model != "")
    {
      d_fifo = fopen(fpga_model.c_str(), "r");
    }
  else
    d_fifo = NULL;
}

/* This models access to the FPGA registers used to configure the part of the
   controls in the fpga */
bool gyro_relay_model_source::_write_fpga_reg(int reg, int value)
{
  switch(reg)
    {
    case FR_RX_MUX: // rx mux
      d_rx_swap = value & 0x10; // could do more with this
      break;
    case FR_MASTER_CTRL: // rx enable, tx enable
      d_rxenable = value & 0x2;
      d_txenable = value & 0x1;
      break;
    case FR_MODE: // loopback
      d_loopback = value & 0x1;
      break;
    case FR_USER_17: // enables
```

```
      d_pll_pid.enable(value & 0x8);
      break;
    case FR_USER_9: // pll filter params
      d_new_pll_a[0] = fixed2float16(value & 0xffff);
      d_new_pll_a[1] = fixed2float16((value >> 16) & 0xffff);
      break;
    case FR_USER_8: // pll filter params
      d_new_pll_b[1] = fixed2float16(value & 0xffff);
      d_pll_updated = true;
      break;
    case FR_RX_FREQ_2: // f0
      d_pll_f0 = value;
      break;
    case FR_USER_20: // pll delay
      d_pll_delay = value;
      break;
    case FR_USER_6: // Ap
      d_pll_pid.set_Ap(fixed2float16(value & 0xffff));
      break;
    case FR_USER_7: // Ai, Ad
      d_pll_pid.set_Ai(fixed2float16(value & 0xffff));
      d_pll_pid.set_Ad(fixed2float16((value >> 16) & 0xffff));
      break;
    case FR_USER_12: // feedthrough
      d_ftc[0] = fixed2float16(value & 0xffff);
      break;
    case FR_USER_13: // feedthrough
      d_ftc[1] = fixed2float16(value & 0xffff);
      break;
    case FR_USER_14: // feedthrough
      d_ftc[2] = fixed2float16(value & 0xffff);
      break;
    case FR_USER_15: // feedthrough
      d_ftc[3] = fixed2float16(value & 0xffff);
      break;
    default:
      break;
    }
  return true;
}

gyro_relay_model_source::~gyro_relay_model_source()
{
  gsl_odeiv_evolve_free (d_evolve);
  gsl_odeiv_control_free (d_control);
  gsl_odeiv_step_free (d_step);
}

bool gyro_relay_model_source::set_mux(int mux)
{
  d_tx_swap = mux & 0x1;
  return true;
}

bool gyro_relay_model_source::set_nchannels(unsigned nchannels)
{
  d_nch = nchannels;
  return true;
}

std::vector<double>
gyro_relay_model_source::monitor()
{
  std::vector<double> a;
  a.push_back(d_rotated*180/M_PI);
  a.push_back(d_irate*180/M_PI);
  a.push_back(0.);
  return a;
```

```
}

bool
gyro_relay_model_source::start()
{
  d_iptr = NULL; // make sure work looks for a new message
  return true;
}

bool
gyro_relay_model_source::stop()
{
  return true;
}

int
gyro_relay_model_source::work (int noutput_items,
                     gr_vector_const_void_star &input_items,
                     gr_vector_void_star &output_items)
{
  /* input is 4 streams of floats interleaved (ix,qx,iy,qy), output is
     8 streams of floats interleaved (cx,sx,cy,sy,lih,lil,lch,lcl */
  float *optr = (float *) output_items[0];

  if(d_fifo != NULL)
    {
      int rp,ip;
      for (int it=0; it < noutput_items/d_nch; it++)
        fscanf(d_fifo, "%x\n", &rp, &ip);
    }
  if(d_iptr == NULL)
    {
      d_msg = d_msgq->delete_head();  // blocking message fetch
      d_iptr= (float *)(d_msg->msg());
      d_ninput_items = d_msg->arg2();
      if(d_msg->arg1() == 2) // done
        {
          fprintf(stderr, "gyro_relay_model_source::work DONE\n");
          return -1;
        }
    }
  // input is 4xCont, output is 4xsense,lih,lil,lch,lcl
  int nprocess = d_ninput_items < noutput_items/d_nch ?
    d_ninput_items : noutput_items/d_nch;
  // need_extra causes an input to be processed to supply a request for a
  // partial output. Do this when we already have an extra but an even
  // number are requested, or don't have an extra and an odd
  // number are requested.
  int need_extra = (d_ninput_items > noutput_items/2) // enough input
    and ( (d_extra.size() and !(noutput_items % d_nch)) // noutput full set
          or (d_extra.size() < (noutput_items % d_nch)) ) ?
    ((noutput_items % d_nch)-d_extra.size()) : 0;
  int processed = 0; // number of output created
  if (d_extra.size()) // use extra first
    {
      int it;
      for (it = 0; it < d_extra.size() and it < noutput_items; it++)
        {
          *optr++ = d_extra.back();
          d_extra.pop_back();
        }
      processed+=it;
    }

  if(d_pll_updated)
    {
      d_pll_filt.set_taps(d_new_pll_a, d_new_pll_b);
      d_pll_updated = false;
    }

  if(d_tuning_update)
    {
      d_Ks = d_new_Ks;
      d_tuning_update = false;
      printf("Loading new Ks");
      printf(": %.5e %.5e %.5e %.5e\n", d_Ks[0], d_Ks[1], d_Ks[2], d_Ks[3]);
    }

  if(d_loopback) // this is a test mode
    {
      fprintf(stderr, "Loopback %d %d\n", nprocess, d_nch);
      for (int it = 0; it < nprocess ; it++)
        {
          *optr++ = *d_iptr++;
          *optr++ = 0;
          d_ninput_items--;
          processed += 2;
        }
      if(d_ninput_items == 0)
        d_iptr = NULL; // used up all input data
      return processed;
    }

  d_t = 0.;
  d_ti = 0.;
  float sense1, sense2, drive1, drive2, qdrive, idrive;
  float cx, sx, cy, sy, cdemod, sdemod;
  float ix, qx, iy, qy;
  float lih,lil,lch,lcl;

  for (int it = 0; it < nprocess/4 ; it++)
    {
      // modulate
      gr_sincosf(d_txphase*FSCALE, &qdrive, &idrive);
      ix = *(d_iptr  );
      qx = *(d_iptr+1);
      iy = *(d_iptr+2);
      qy = *(d_iptr+3);
      drive1 = ix*idrive+qx*qdrive;
      drive2 = iy*idrive+qy*qdrive;
      d_iptr+=4;
      d_ninput_items-=4;
      if(d_tx_swap)
        solve_1step(drive2, drive1, sense1, sense2);
      else
        solve_1step(drive1, drive2, sense1, sense2);
      gr_sincosf(d_rxphase*FSCALE, &sdemod, &cdemod);
      if(d_rx_swap)
        {
          cx = sense2 * cdemod;
          sx = sense2 * sdemod;
          cy = sense1 * cdemod;
          sy = sense1 * sdemod;
        }
      else
        {
          cx = sense1 * cdemod;
          sx = sense1 * sdemod;
          cy = sense2 * cdemod;
          sy = sense2 * sdemod;
        }
      pll_1step(cx,sx,cy,sy);
      *optr++ = cx; *optr++ = sx; *optr++ = cy; *optr++ = sy;
      //if d_nch > 2 {}
      lih = floor(d_li/TWO16);
      lil = fmod( d_li,TWO16);
```

```
      lch = floor(d_lc/TWO16);
      lcl = fmod( d_lc,TWO16);
      *optr++ = lih; *optr++ = lil;
      *optr++ = lch; *optr++ = lcl;
      processed += 8;
    }

  if (need_extra) // odd, but enough data for all
    {
      // modulate
      gr_sincosf(FSCALE*d_txphase, &qdrive, &idrive);
      drive1 = *(d_iptr  )*idrive+*(d_iptr+1)*qdrive;
      drive2 = *(d_iptr+2)*idrive+*(d_iptr+3)*qdrive;
      d_iptr+=4;
      d_ninput_items-=4;
      if(d_tx_swap)
        solve_1step(drive2, drive1, sense1, sense2);
      else
        solve_1step(drive1, drive2, sense1, sense2);
      gr_sincosf(FSCALE*d_rxphase, &sdemod, &cdemod);
      if(d_rx_swap)
        {
          cx = sense2 * cdemod;
          sx = sense2 * sdemod;
          cy = sense1 * cdemod;
          sy = sense1 * sdemod;
        }
      else
        {
          cx = sense1 * cdemod;
          sx = sense1 * sdemod;
          cy = sense2 * cdemod;
          sy = sense2 * sdemod;
        }
      pll_1step(cx,sx,cy,sy);
      lih = floor(d_li/TWO16);
      lil = fmod(d_li, TWO16);
      lch = floor(d_lc/TWO16);
      lcl = fmod(d_lc, TWO16);
      d_extra.push_back(lcl);
      d_extra.push_back(lch);
      d_extra.push_back(lil);
      d_extra.push_back(lih);
      d_extra.push_back(sy);
      d_extra.push_back(cy);
      d_extra.push_back(sx);
      d_extra.push_back(cx);
      for(int it = 0; it < need_extra; it++)
        {
          *optr++ = d_extra.back();
          d_extra.pop_back();
        }
      processed+=need_extra;
    }
  if(d_ninput_items == 0)
    {
      d_iptr = NULL; // used up all input data
    }
  return processed;
}

inline void
gyro_relay_model_source::pll_1step(float cx, float sx, float cy, float sy)
{
  double li, lc, lp_lc;
  long rd_lc;
  li = floor(d_pll_filt.filter((cx*sx + cy*sy)/TWO16));
  lc = d_pll_pid.update(d_li);
  lp_lc = d_pll_filt.filter(lc);
  rd_lc = lround(lp_lc);
  d_li = li;
  d_lc = rd_lc;
  d_txphase = (d_txphase + d_pll_f0 + d_lc) & 0xffffffff;
  d_rxphase = (d_txphase + d_pll_delay) & 0xffffffff;
}

void gyro_relay_model_source::set(double rate, double accel)
{
  d_rate = rate*M_PI/180;
  d_rate_accel = accel*M_PI/180;
}

void gyro_relay_model_source::set_rate_freq(double rate_freq, double accel)
{
  d_rate_freq = M_TWOPI*rate_freq; // convert to radians
  d_rate_accel = accel*M_PI/180;
}

inline void
gyro_relay_model_source::solve_1step(float drive1, float drive2,
                                  float &sense1, float &sense2)
{
  std::vector<double> ft(2);
  double rotations, rs, rc, rate, last_t;

  d_ti += 1/d_signal_rate;
  d_drive1 = drive1;
  d_drive2 = drive2;
  while (d_t < d_ti) // use variable step-size between ti
    {
      d_rate_phase = fmod(d_rate_phase+d_rate_freq*d_dt,M_TWOPI);
      sincos(d_rate_phase, &rs, &rc);

      rate = fabs(d_rate*rc-d_irate) < (d_rate_accel*d_dt) ?
        d_rate*rc : d_irate+sgn(d_rate*rc-d_irate)*d_rate_accel*d_dt;
      d_drate = (rate-d_irate)/d_dt;
      d_irate = rate;
      last_t = d_t;
      d_err = gsl_odeiv_evolve_apply (d_evolve, d_control, d_step,
                                      &d_sys,
                                      &d_t, d_ti, &d_tolerance,
                                      d_state);
      d_dt = d_t - last_t;
    }
  d_rotated = modf((d_rotated + d_irate/d_signal_rate), &rotations);
  d_pp = d_pp*(1-1/128.)+drive1*d_state[0]/128;
  d_pv = d_pv*(1-1/128.)+drive1*d_state[1]/128;
  // Feed through calculation, real is x, imag is y
  ft[0] = d_gain[0]*(d_ftg[0]*(drive1 - d_ld1) + d_ftg[2]*(drive2 - d_ld2));
  ft[1] = d_gain[1]*(d_ftg[1]*(drive1 - d_ld1) + d_ftg[3]*(drive2 - d_ld2));
#ifdef SENSE_ROTATED
#ifdef POSITION  /* position readout */
  sense1 = ft[0]+(float)(d_gain[0]*(d_state[0]*d_CthS
                                    - d_state[2]*d_SthS)/USRP_SCALE);
  sense2 = ft[1]+(float)(d_gain[1]*(d_state[0]*d_SthS
                                    + d_state[2]*d_CthS)/USRP_SCALE);
#else           /* velocity readout */
  sense1 = ft[0]+(float)(d_gain[0]*(d_state[1]*d_CthS
                                    - d_state[3]*d_SthS)/USRP_SCALE/d_vs);
  sense2 = ft[1]+(float)(d_gain[1]*(d_state[1]*d_SthS
                                    + d_state[3]*d_CthS)/USRP_SCALE/d_vs);
#endif // position
#else
#ifdef POSITION /* position readout */
  sense1 = ft[0]+(float)(d_gain[0]*(d_state[0]/USRP_SCALE));
  sense2 = ft[1]+(float)(d_gain[1]*(d_state[2]/USRP_SCALE));
```

```
#else           /* velocity readout */
  sense1 = ft[0]+(float)(d_gain[0]*(d_state[1]/USRP_SCALE/d_vs));
  sense2 = ft[1]+(float)(d_gain[1]*(d_state[3]/USRP_SCALE/d_vs));
#endif // position
#endif // rotated
  d_ld1 = drive1;
  d_ld2 = drive2;
}

int relay_func (double t, const double y[], double f[],
          void *params)
{ // model from Lynch "Gyro analysis by method of averaging"
  // Y[0:3] = x vx y vy, f[0:3] = dx dvx dy dvy
  gyro_relay_model_source *m = (gyro_relay_model_source *)params;
  // dx/dt = vx
  f[0] = y[1];
  // dvx/dt = fx + ...
  f[1] = m->d_drive1*m->d_gain[2]*30                   // electrical forcing
    + 2*m->d_kr*m->d_irate*y[3]                        // coriolis force
    + m->d_kr*m->d_drate*y[2]                          // angular acceleration
    - m->d_dmp * y[1]                                  // damping
    - m->d_ddp * (y[1] * m->d_rtc + y[3] * m->d_rts)  // damping mismatch
    - y[0] * (m->d_w0 - m->d_kc*m->d_irate*m->d_irate)// spring + centripedal
    + m->d_wdw * (y[0] * m->d_rwc + y[2] * m->d_rws)  // frequency mismatch
    + m->d_Ks[0] * y[0] + m->d_Ks[1] * y[2];          // electrostatic tuning
  // dy/dt = vy
  f[2] = y[3];
  // dvy/dt = fy - ...
  f[3] = m->d_drive2*m->d_gain[3]*30                  // electrical forcing
    - 2*m->d_kr*m->d_irate*y[1]                        // coriolis force
    - m->d_kr*m->d_drate*y[0]                          // angular acceleration
    - m->d_dmp * y[3]                                  // damping
    + m->d_ddp * (y[3] * m->d_rtc - y[1] * m->d_rts)  // damping mismatch
    - y[2] * (m->d_w0 - m->d_kc*m->d_irate*m->d_irate)// spring + centripedal
    - m->d_wdw * (y[2] * m->d_rwc - y[0] * m->d_rws)  // frequency mismatch
    + m->d_Ks[2] * y[0] + m->d_Ks[3] * y[2];          // electrostatic tuning
  return GSL_SUCCESS;
}
```

APPENDIX B

Hybrid Mode Gyroscope Control Code

```
#include <gyro_common.h>
#include <gyro_wa_control_relay_2.h>
#include <gr_io_signature.h>
#include <gr_sincos.h>
#include <math.h>
#include <gr_math.h>
#include <stdio.h>
#include <string.h>

//#define SQUARE_DRIVE
#define VELOCITY -1 // -1 for velocity, 1 for position

gyro_wa_control_relay_2::gyro_wa_control_relay_2
(                                                    \
 bool velocity, float lock_thresh, const std::vector<double> &dm_ff_taps,
 const std::vector<double> &dm_fb_taps, float x_gain, float y_gain,
 float clip, float elock, float qlock, float slock, float smp_rate)
  : gr_sync_block ("wa_control_relay_2",
                   gr_make_io_signature (4, 4, sizeof (float)),
                   gr_make_io_signature (16, 16, sizeof (float))),
    d_amp(0.), d_err(0.), d_lock_thresh(lock_thresh), d_pll_enb(false),
    d_theta_offset(0.0), d_theta_last(0.), d_ext_enb(false),
    d_filt_cx(dm_ff_taps, dm_fb_taps), d_filt_sx(dm_ff_taps, dm_fb_taps),
    d_filt_cy(dm_ff_taps, dm_fb_taps), d_filt_sy(dm_ff_taps, dm_fb_taps),
    d_rx_gain(x_gain), d_ry_gain(y_gain), d_dm_updated(false),
    d_epid(clip,-clip,elock), d_qpid(clip,0.,qlock), d_spid(clip,0.,slock),
    d_dxpid(clip,0,elock),d_dypid(clip,0,elock),
    d_fxpid(clip,0,qlock),d_fypid(clip,0,qlock),
    d_dpid(15,0,1), d_count(0), d_delay(0),
    d_dd_locked(false), d_dd_enable(false),
    d_smp_rate(smp_rate), d_Kerr(0.), d_Ferr(0.), d_Kq(0.), d_plus_pi(M_PI)
{
  d_v = 0;
}

void
gyro_wa_control_relay_2::set_dm_taps (const std::vector<double> &ff_taps,
                                      const std::vector<double> &fb_taps)
{
  d_new_dm_ff_taps = ff_taps;
  d_new_dm_fb_taps = fb_taps;
  d_dm_updated  = true;
}

/* Check whether tha PLL is locked, used to control initialization sequence */
bool
gyro_wa_control_relay_2::locked()
{
  bool locked = (d_amp > 100) && (fabs(d_err/d_amp) < d_lock_thresh);
  if(locked)
    d_ext_enb = true; // this won't unset if 'lock' is lost
  return locked;
}

void
gyro_wa_control_relay_2::enable(bool enb)
{
  d_pll_enb  = enb;
}

gyro_wa_control_relay_2::~gyro_wa_control_relay_2()
{
}

/* Detect the delay for feed-forward correction of the delay */
int
gyro_wa_control_relay_2::delay_detect (int noutput_items,
                                       gr_vector_const_void_star &input_items,
                                       gr_vector_void_star &output_items)
{
  float *ix = (float *) output_items[0]; // in phase x axis mod
  float *qx = (float *) output_items[1]; // out of phase x axis mod
  float *iy = (float *) output_items[2]; // in phase y axis mod
  float *qy = (float *) output_items[3]; // out of phase y axis mod
  float *p_cx = (float *)input_items[0];
  float *p_sx = (float *)input_items[1];
  float *p_cy = (float *)input_items[2];
  float *p_sy = (float *)input_items[3];

  if(d_dd_enable && (!d_delay) && (d_count < 16384))
    { // this is over 50ms, skip once d_delay is set
      for(int it=0; it<noutput_items; it++)
        {
          *ix++ = d_count++;
        }
    }
  else // don't set locked until the output is cleared
    {
      if(d_dd_enable)
      {
        if(!d_delay) // only set this once
          {
            d_delay = d_count - floorf(*p_cx);
            d_tmp_delay = d_delay;
          }
        d_count -= noutput_items;
        if(d_count <= 0)
          { // send out as many as were already sent to clear out the queue
           d_dd_locked = true;
          }
      }
      memset(ix, 0, noutput_items*sizeof(float));
    }
  memset(qx, 0, noutput_items*sizeof(float));
  memset(qy, 0, noutput_items*sizeof(float));
  memset(iy, 0, noutput_items*sizeof(float));
  return noutput_items;
}
```

```
void
gyro_wa_control_relay_2::dd_enable(bool enb)
{
  if(enb)
    d_delay = d_tmp_delay;
  else
    d_delay = 0;
  d_dd_enable = enb;
}

void gyro_wa_control_relay_2::dd_clear()
{
  d_dd_locked = false;
  d_count = 0;
  d_delay = 0;
}

int
gyro_wa_control_relay_2::work (int noutput_items,
                               gr_vector_const_void_star &input_items,
                               gr_vector_void_star &output_items)
{
  if (d_dm_updated)
    {
      d_filt_cx.set_taps(d_new_dm_ff_taps,d_new_dm_fb_taps);
      d_filt_sx.set_taps(d_new_dm_ff_taps,d_new_dm_fb_taps);
      d_filt_cy.set_taps(d_new_dm_ff_taps,d_new_dm_fb_taps);
      d_filt_sy.set_taps(d_new_dm_ff_taps,d_new_dm_fb_taps);
      d_dm_updated = false;
      return 0;                // history requirements may have changed.
    }
  float *ix = (float *) output_items[0]; // in phase x axis mod
  float *qx = (float *) output_items[1]; // out of phase x axis mod
  float *iy = (float *) output_items[2]; // in phase y axis mod
  float *qy = (float *) output_items[3]; // out of phase y axis mod
  float *E  = (float*) output_items[4];
  float *Q  = (float*) output_items[5];
  float *Th = (float*) output_items[6];
  float *Li = (float*) output_items[7];
  float *epid=(float*) output_items[8];
  float *qpid=(float*) output_items[9];
  float *spid=(float*) output_items[10];
  float *rate=(float*) output_items[11];
  float *fmmx=(float*) output_items[12];
  float *fmmy=(float*) output_items[13];
  float *dmmx=(float*) output_items[14];
  float *dmmy=(float*) output_items[15];

  float *p_cx = (float *)input_items[0];
  float *p_sx = (float *)input_items[1];
  float *p_cy = (float *)input_items[2];
  float *p_sy = (float *)input_items[3];

  int   size = noutput_items;
  float cx, sx, cy, sy;
  float cx2,sx2,cy2,sy2;
  float S, R; // ~x,y components of primary oscillation
  float sint, cost, sin2t, cos2t; // sin and cos of orientation, for efficiency
  float ffth; // fed-forward angle

  if(!d_pll_enb) // just send 0
    {
      delay_detect(noutput_items, input_items, output_items);
      memset(E, 0, noutput_items*sizeof(float));
      memset(Q, 0, noutput_items*sizeof(float));
      memset(Th,0, noutput_items*sizeof(float));
      memset(epid, 0, noutput_items*sizeof(float));
      memset(qpid, 0, noutput_items*sizeof(float));
      memset(spid, 0, noutput_items*sizeof(float));
      memset(rate, 0, noutput_items*sizeof(float));
      return noutput_items;
    }

  while (size-- > 0)
    { // XX_FILTER based on filter and invert cX based on velocity
      cx = CX_FILTER(*p_cx++ * VELOCITY * d_rx_gain);
      sx = SX_FILTER(*p_sx++ * d_rx_gain);
      cy = CY_FILTER(*p_cy++ * VELOCITY * d_ry_gain);
      sy = SY_FILTER(*p_sy++ * d_ry_gain);
      cx2 = cx*cx;
      sx2 = sx*sx;
      cy2 = cy*cy;
      sy2 = sy*sy;
      // Extract control parameters
      *E = (cx2+sx2+cy2+sy2)*USRP_SCALE;
      *Q = 2*(cx * sy - cy * sx)*USRP_SCALE;
      R  = (cx2+sx2-cy2-sy2)*USRP_SCALE;
      S  = 2*(cx * cy + sx * sy)*USRP_SCALE;
      d_err = (2*(cx * sx + cy * sy))*USRP_SCALE; // Li

      // Orientation Expansion
      *Th = ((S == 0.0) && (R == 0.0)) ? 0 // avoid NaN, only happens at startup
        : (atan(S/R) // avoid NaN, [-pi/2,+pi/2]
           + (R > 0 ? (cx > 0 ? 0 : (S > 0 ? -M_TWOPI : M_TWOPI)) // R > 0
           : d_plus_pi)); // R < 0, [-2pi,+2pi]
      d_plus_pi = *Th > 0 ? M_PI : -M_PI;
      // PIDS
      *epid = d_epid.update(*E,1);  // average damping compensation
      *qpid = d_qpid.update(*Q,1);  // quadrature compensation
      *spid = d_spid.update(*Th);   // steering control
      d_rate = fmod(*Th-d_theta_last,M_PI/2);
      d_rate = fabs(d_rate) > 26.17/d_smp_rate ? 0 : d_rate;

      cos2t = cos(*Th); sin2t = sin(*Th);
      *dmmx  =  d_dxpid.update(*E,-d_rate*d_smp_rate*sin2t);
      *dmmy  =  d_dypid.update(*E, d_rate*d_smp_rate*cos2t);
      /* this oscillates around set point if gain is higher than ampl gain
      // fails when rate is negative?
      *dmmx  =  d_dxpid.update(*E, sin2t);
      *dmmy  =  d_dypid.update(*E,-cos2t); */
      /* this the one from theory, settles slowly but works
      *dmmx  =  d_dxpid.update(*E, cos2t);
      *dmmy  =  d_dypid.update(*E, sin2t); */

      /* This works without fmm in steering, with Ai,Ad == 0in qpid,
         but takes long time to settle
      *fmmx  =  d_fxpid.update(*Q, cos2t);
      *fmmy  =  d_fypid.update(*Q, sin2t);
      */
      /* This works with Ai,Ad != 0, doesn't oscillate when settling.
         Problem with negatve rates
      *fmmx  =  d_fxpid.update(*Q, cos2t);
      *fmmy  =  d_fypid.update(*Q, sin2t); */
      /* This works with Ai,Ad != 0, doesn't oscillate when settling. */
      *fmmx  =  d_fxpid.update(*Q, -d_rate*d_smp_rate*cos2t);
      *fmmy  =  d_fypid.update(*Q, -d_rate*d_smp_rate*sin2t);
      d_theta_last = *Th;
      // Calculate scaling of drive signals including feed forward
      // Next Th is Th+rate*d_delay (so instantaneous Th is correct)
      ffth = *Th + d_rate * d_delay; // feed forward orientation
      if(!d_epid.enable())
        {
          cost = 0.5;
          sint = 0.5;
        }
      else
```

```
        {
          sint = sin(ffth/2); cost = cos(ffth/2);
        }
      *ix = d_tx_gain*((*epid+*dmmx) * cost - *spid*sint + *dmmy*sint);
      *iy = d_ty_gain*((*epid-*dmmx) * sint + *spid*cost + *dmmy*cost);
      *qx = d_tx_gain*( *qpid*sint - *fmmx*cost - *fmmy*sint);
      *qy = d_ty_gain*(-*qpid*cost + *fmmx*sint - *fmmy*cost);
      *rate = d_rate * d_smp_rate;
      *Li = d_err;
      d_v++;
      ix++; iy++; qx++; qy++;
      E++; Q++; Th++; epid++; qpid++; spid++; rate++; Li++;
      fmmx++; fmmy++; dmmx++; dmmy++;
    }
  // calculate values for tuning bias voltages.  These are more epxensive
  // and only need to be done very slowly, so just do it once per buffer
  // This is 2(Theta_w)
  d_Kerr = atan2(S,R) - atan2(*(qy-1), *(qx-1));
  // This is delta_w * tau
  d_Ferr = *(iy-1)+*(ix-1) ?
    (*(qy-1)-*(qx-1))/(*(iy-1)+*(ix-1)) : *(qy-1)/(*(ix-1));
  d_amp = *(E-1);
  return noutput_items;
}
```

APPENDIX C

Rate Gyroscope Firmware

```
//  Pipelined gyroscope parameter extractor and controller
module rate_gyro_cont (clk, reset, enable,
                       serial_addr, serial_data, serial_strobe,
                       cx, sx, cy, sy, interp_strobe,
                       interp_rate, strobe,
                       ix, iy, qy, qx, phase);
        parameter SET_ADDR = 0;
   parameter FREQ_ADDR = 0;
        parameter CONT_ADDR = 0;
   parameter resolution= 32;
   parameter LSS = 1; // Lsum scale
        parameter LCS = 0; // Lcont scale

   input wire    clk, reset, enable, strobe;
   input wire [15:0] cx;
   input wire [15:0] sx;
   input wire [15:0] cy;
   input wire [15:0] sy;
   input wire        interp_strobe;
   input wire [7:0]  interp_rate;

   input wire [6:0]  serial_addr;
   input wire [31:0] serial_data;
   input wire        serial_strobe;

   output wire [31:0] ix;
   output wire [31:0] iy;
   output wire [31:0] qy;
   output wire [31:0] qx;
   output reg  [31:0] phase;

   reg [3:0]     pipeline;
   reg           rdy; // finished one sample, wait for another
   reg [1:0]     pid_mux, iir_mux;
   reg           pid_strobe, iir_strobe;
   wire [31:0]   pid_sum, pid_last, pid_cont,
                 pid_new_sum, pid_new_last;
   wire [15:0]   pid_Ap, pid_Ai, pid_Ad;
   wire [15:0]   iir_in, iir_in_hist;
   wire [31:0]   iir_out, iir_out_hist;
   wire [15:0]   iir_b1, iir_a0, iir_a1;
```

```
    wire [31:0]   iir_out_limited;
    reg [31:0]    pid_err;
    reg [15:0]    Eerr, Eerr_last; // inputs to IIR
    reg [31:0]    Esum, Elast, Econt;
    reg [31:0]    E_iir_out_hist,L_iir_out_hist,S_iir_out_hist,Q_iir_out_hist;
    reg [31:0]    Qsum, Qlast, Qcont;
    reg [31:0]    Ssum, Slast, Scont;
    reg [31:0]    Lsum, Llast, Lcont;
    wire [15:0]   Einit;//, Qinit, Sinit;
    wire [31:0]   freq0;
    wire [31:0]   cont_config; // cont_config[3:0] == enable spid,qpid,lpid,epid
    wire [15:0]   Eset;

    reg [15:0] cx_n, cx_n1; // latched input and prev_input
    reg [15:0] sx_n, sx_n1;
    reg [15:0] cy_n, cy_n1;
    reg [15:0] sy_n, sy_n1;

    assign qx = Lcont;
    assign ix = Econt;
    assign iy = Scont;
    assign qy = Qcont;

    pipelined_pid pid
          ( .clk(clk), .reset(reset), .enable(enable), .strobe(pid_strobe),
        .err(pid_err), .prev_sum(pid_sum), .prev_in(pid_last),
        .new_in(pid_new_last), .new_sum(pid_new_sum), .cont(pid_cont),
        .rdy(pid_rdy), .Ap(pid_Ap), .Ai(pid_Ai), .Ad(pid_Ad) );

    pipelined_iir1 iir
      ( .clk(clk), .reset(reset), .enable(enable), .strobe(iir_strobe),
        .in(iir_in), .out(iir_out), .rdy(iir_rdy),
        .in_hist(iir_in_hist),.out_hist(iir_out_hist),
        .b1(iir_b1),.a0(iir_a0),.a1(iir_a1) );

    rate_gyro_gains rgg
      ( .clk(clk), .pid_mux(pid_mux), .iir_mux(iir_mux),
        .b1(iir_b1), .a0(iir_a0), .a1(iir_a1),
        .Ap(pid_Ap), .Ai(pid_Ai), .Ad(pid_Ad),
        .serial_addr(serial_addr), .serial_data(serial_data),
        .serial_strobe(serial_strobe) );

    assign iir_out_limited = iir_out;
    assign pid_sum  = pid_mux[1] ? ( pid_mux[0] ? Esum : Ssum )
      : (pid_mux[0] ? Qsum : Lsum);
    assign pid_last = pid_mux[1] ? ( pid_mux[0] ? Elast : Slast )
      : (pid_mux[0] ? Qlast : Llast);
    assign iir_in   = iir_mux[1] ? ( iir_mux[0] ? Eerr : cy_n )
      : (iir_mux[0] ? sy_n : sx_n);
    assign iir_in_hist = iir_mux[1] ? ( iir_mux[0] ? Eerr_last : cy_n1 )
      : (iir_mux[0] ? sy_n1 : sx_n1);
    assign iir_out_hist = iir_mux[1] ? ( iir_mux[0] ? E_iir_out_hist : S_iir_out_hist
        )
      : (iir_mux[0] ? Q_iir_out_hist : L_iir_out_hist);

    setting_reg #(SET_ADDR) sr_eset(.clock(clk),.reset(1'b0),.strobe(serial_strobe),.
        addr(serial_addr),.in(serial_data),.out({Einit,Eset}));
    // Phase accumulator for PLL
    setting_reg #(FREQ_ADDR) sr_freq0(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
        ,.addr(serial_addr),.in(serial_data),.out(freq0));
         setting_reg #(CONT_ADDR) sr_config(.clock(clk),.reset(1'b0),.strobe(
             serial_strobe),.addr(serial_addr),.in(serial_data),.out(cont_config));

         always @(posedge clk)
                 if (reset)
                         phase <= #1 0;
`ifdef SLOW_PHASE_UPDATE
                 else if (enable & strobe)
```

```
`else
                else
`endif
                phase <= #1 phase + freq0 + {{LCS{Lcont[31]}},Lcont[31:LCS]};

   always @(posedge clk)
         if (reset)
           begin
          rdy <= #1 1'b1;
                  pipeline <= #1 4'd0;           pid_mux <= #1 2'd0;
          iir_mux <= #1 2'd0;             pid_strobe <= #1 1'd0;
          iir_strobe <= #1 1'd0;          pid_err <= #1 32'd0;
          Eerr <= #1 16'd0;               Eerr_last <= #1 16'd0;
          Esum <= #1 32'd0;               Elast <= #1 32'd0;
          Econt <= #1 32'd0;              E_iir_out_hist <= #1 32'd0;
          L_iir_out_hist <= #1 32'd0;     S_iir_out_hist <= #1 32'd0;
          Q_iir_out_hist <= #1 32'd0;     Qsum <= #1 32'd0;
          Qlast <= #1 32'd0;              Qcont <= #1 32'd0;
          Ssum <= #1 32'd0;               Slast <= #1 32'd0;
          Scont <= #1 32'd0;              Lsum <= #1 32'd0;
          Llast <= #1 32'd0;              Lcont <= #1 32'd0;
          cx_n <= #1 16'd0;               cx_n1 <= #1 16'd0;
          sx_n <= #1 16'd0;               sx_n1 <= #1 16'd0;
          cy_n <= #1 16'd0;               cy_n1 <= #1 16'd0;
          sy_n <= #1 16'd0;               sy_n1 <= #1 16'd0;
           end
         else if (enable & strobe)
           begin
          rdy <= #1 1'b0;
              pipeline<= #1 4'd0;
          pid_mux <= #1 2'h0;
          iir_mux <= #1 2'h0;
          cx_n <= #1 cx; // sample inputs
          sx_n <= #1 sx;
          cy_n <= #1 cy;
          sy_n <= #1 sy;
          cx_n1 <= #1 cx_n; // input history for IIR
          sx_n1 <= #1 sx_n;
          cy_n1 <= #1 cy_n;
          sy_n1 <= #1 sy_n;
          // stobe kickstarts new IIR sequence, rest are driven by rdy
          iir_strobe <= #1 1'b1;
       end
     else if (!rdy && enable && (iir_rdy | pid_rdy))
       begin
          if(iir_rdy) // mux++ & latch history of IIR to correct stream
            begin
               iir_mux <= #1 iir_mux + 1'b1;
                                         pid_err <= #1 iir_out_limited;
               //pid_err <= #1 iir_out; // IIR output always goes to PID input
               case (pipeline)
                 4'd0: iir_strobe <= #1 1'b1;
                 4'd1: iir_strobe <= #1 1'b1;
                 4'd2: iir_strobe <= #1 1'b1;
                 4'd3: iir_strobe <= #1 1'b0; // wait for new data
               endcase
               case (pipeline)
                 4'd0:
                   begin
                      // stobe kickstarts new PID sequence, then driven by rdy
                      pid_strobe <= #1 1'b1;
                      L_iir_out_hist <= #1 iir_out_limited;
                      Eerr <= #1 cx_n - Eset;
                      Eerr_last <= #1 Eerr;
                   end
                 4'd1: Q_iir_out_hist <= #1 iir_out_limited;
                 4'd2: S_iir_out_hist <= #1 iir_out_limited;
                 4'd3: E_iir_out_hist <= #1 iir_out_limited;
                endcase // case (pipeline)
             end // if (iir_rdy)
           else
             iir_strobe <= #1 1'b0;
           if(pid_rdy) // mux++ & latch outputs of PID to correct stream
             begin
                pid_mux <= #1 pid_mux + 1'b1;
                case (pipeline)
                  4'd1: pid_strobe <= #1 1'b1;
                  4'd2: pid_strobe <= #1 1'b1;
                  4'd3: pid_strobe <= #1 1'b1;
                  4'd4:
                    begin
                       pid_strobe <= #1 1'b0; // wait for new data
                       rdy <= #1 1'b1;
                    end
                endcase
                case (pipeline)
                  4'd1: Lcont <= #1 pid_cont; //{{3{pid_cont[31]}},pid_cont[31:3]};
                  4'd2: Qcont <= #1 pid_cont;
                  4'd3: Scont <= #1 pid_cont;
                  4'd4: Econt <= #1 pid_cont;
                endcase // case (pipeline)
                case (pipeline)
                                          4'd1: Lsum <= #1 cont_config[3] ? {{LSS{
                                              pid_new_sum[31]}},pid_new_sum[31:LSS]}
                                              : 32'b0;
                  4'd2: Qsum <= #1 cont_config[2] ? pid_new_sum : 32'b0;
                  4'd3: Ssum <= #1 cont_config[1] ? pid_new_sum : 32'b0;
                  4'd4: Esum <= #1 cont_config[0] ? pid_new_sum : {Einit,16'b0};
                endcase // case (pipeline)
                case (pipeline)
                  4'd1: Llast <= #1 pid_new_last;
                  4'd2: Qlast <= #1 pid_new_last;
                  4'd3: Slast <= #1 pid_new_last;
                  4'd4: Elast <= #1 pid_new_last;
                endcase // case (pipeline)
             end // if (pid_rdy)
           else if (pipeline != 4'd0)
             pid_strobe <= #1 1'b0;
           if(iir_rdy && (pid_rdy | pipeline == 0))
             pipeline <= #1 pipeline + 1'b1;
        end // if (!rdy && enable && (iir_rdy | pid_rdy))
      else // turn off strobes
        begin
           iir_strobe <= #1 1'b0;
           pid_strobe <= #1 1'b0;
        end
endmodule // rate_gyro_cont

// This is first order filter for applications where flatness isn't critical
module pipelined_iir1 (clk, reset, enable, strobe,
                                 in, out, rdy, in_hist, out_hist, b1, a0, a1);
   parameter RES       = 32;

   input wire clk, reset, enable, strobe;
   output reg rdy;
   input wire [15:0]  in; // sensed signals
   input wire [15:0]  in_hist; // input history for this stream
   input wire [31:0]  out_hist; // out history for this stream
   output reg [31:0]  out;
   input wire [15:0]  b1,a0,a1;

   wire [15:0]        coeff; // muxed coeff
   wire [15:0]        val;   // muxed memory
   wire [31:0]        sum;
   wire [30:0]        prod;
   wire [31:0]        prod_e;
wire [31:0]         pre_acc;
reg [31:0]          acc;
reg [1:0]           mux;
reg signed [15:0]   in_n;
reg signed [15:0]   in_n1;
reg signed [31:0]   out_n; // need full resolution to support low freqs
reg signed [15:0]   b1_n, a0_n, a1_n; // latched gains
reg                 set;

assign prod_e = {prod,prod[30]};
assign sum = acc + prod_e;
// pre_acc is remainder of out that would otherwise be discarded
// this allows small positive signals to grow even if the initial output is 0
// negative numbers tend to be overcounted (negaive anything rounds
// to at least - 1), so the postivie remainder slows signal growth and
// makes the filter balanced for +/- signals
assign pre_acc = {16'b0,out_n[15:0]};

mult mult(.clock(clk),.x(val),.y(coeff),
          .product(prod),.enable_in(1'b1),
          .enable_out());
// y[n] = a[0]*x[n]+a[1]*x[n-1]+b[1]*y[n-1]
// do a[1] then a[0] then b[1]
assign coeff = mux[1] ? b1_n : (mux[0] ? a1_n : a0_n);
// when mux == 0, in_n is previous input,
// when mux == 1, in_n will be the newly sampled input
assign val   = mux[1] ? out_n[31:16] : (mux[0] ? in_n : in_n1);

always @(posedge clk)
      begin
     if(reset)
       begin
          out    <= #1 32'b0;              out_n  <= #1 32'b0;
          rdy    <= #1 1'b0;               in_n   <= #1 16'd0;
          in_n1  <= #1 16'd0;              acc    <= #1 32'd0;
          mux    <= #1 2'd0;               a0_n   <= #1 16'd0;
          a1_n   <= #1 16'd0;              b1_n   <= #1 16'd0;
          set    <= #1 1'b0;
               end
     else
       begin
          if(strobe & enable)
            begin
               in_n1  <= #1 in_hist;
               in_n   <= #1 in;
               out_n  <= #1 out_hist;
               acc    <= #1 32'd0;
               rdy    <= #1 1'b0;
               mux    <= #1 1'b0;
               a0_n   <= #1 a0;
               a1_n   <= #1 a1;
               b1_n   <= #1 b1;
               set    <= #1 1'b1;
            end
          else if (set && !rdy)
            begin
               mux <= #1 mux + 2'b1;
               case(mux)
                 2'd0 : acc <= #1 pre_acc;
                 2'd1 : acc <= #1 sum; // acc += a0*prev_in
                 2'd2 : acc <= #1 sum; // acc += a1*in
                 2'd3 :
                   begin
                      set <= #1 1'b0; // disable mux (set by strobe)
                      rdy <= #1 1'b1; // strobe ready to parent module
                      out <= #1 sum;  // acc += b1*prev_out
                   end
               endcase // case (mux)
```

```
              end // else: !if((strobe | rdy) & enable)
            else
              rdy <= #1 1'b0;
         end // else: !if(reset)
    end // always @ (posedge clk)
endmodule // pipelined_iir1

// PID which rounds by only using top 16 bits.  Should use pipelined_pid16, but
// this is the version used for the data presented here.
module pipelined_pid (clk, reset, enable, strobe,
                        err, prev_sum, prev_in, new_in, new_sum, cont, rdy,
                        Ap, Ai, Ad);
   parameter RES       = 32;
   parameter SCALE     = 5;
   parameter ISCALE    = 0;
   parameter DSCALE    = 0;
   input wire clk;
   input wire reset;
   input wire enable;
   input wire strobe;

   input wire [RES-1:0] err; // this is the input to control
   input wire [RES-1:0] prev_sum;
   input wire [RES-1:0] prev_in;
   output reg [RES-1:0] cont;
   output reg [RES-1:0] new_sum;
   output reg [RES-1:0] new_in;
   output reg           rdy;
   input wire [15:0]    Ap,Ai,Ad;

   wire [15:0]  gain;  // muxed gain
   wire [15:0]  val;   // muxed memory
   wire [31:0]  sum;
   wire [31:0]  diff;
   wire [31:0]  diffl;
   wire [30:0]  prod;
   wire [31:0]  prod_e;
   //wire [15:0]  out_short;
   reg [31:0]   acc;
   reg [2:0]    mux;
   reg [15:0]   Ap_n,Ai_n,Ad_n;
   reg signed [RES-1:0] in_n;
   reg signed [RES-1:0] in_n1;
   reg               set;
   wire [15:0]       err_short;
   assign err_short = err[31:16];

   mult mult(.clock(clk),.x(val),.y(gain),
            .product(prod),.enable_in(1'b1),
            .enable_out());
   // y[n] = pg*x[n] + sum
   // sum  = sum[n-1] + pi*x[n] + pd*(x[n]-x[n-1])
   assign gain = mux[1] ? Ap_n : (mux[0] ? Ad_n : Ai_n);
   // when mux == 0, in_n is previous input,
   // when mux == 1, in_n will be the newly sampled input
   assign diff  = in_n - in_n1;
   assign diffl = (diff[31] & (diff > 32'hffff0000)) ? 32'h0 : diff;
   //assign diffl = (diff[31] & (diff < 32'hffff8000)) ? 32'h80000000 : (~diff[31] &
       (diff > 32'h00007fff) ? 32'h7fff0000 : {diff[15:0],16'b0});
   assign val   = mux[1] ? in_n[31:16] : (mux[0] ? {{DSCALE{diffl[31]}},diffl[31:(
       DSCALE+16)]}  : in_n[31:16]);
   //assign val   = mux[1] ? in_n[31:16] : (mux[0] ? diffl[(31-DSCALE):(16-DSCALE)]
            : in_n[31:16]);
   // scaling for sample rat, mux is +1 from gain and val. Use same scale for P and D
        gains
   assign prod_e = mux[1] ? {prod[30],prod[30-SCALE:0],{SCALE{1'b0}}} : {{1+ISCALE{
       prod[30]}},prod[30:ISCALE]};
   assign sum = acc + prod_e;

   always @(posedge clk)
     if(reset)
       begin
          cont    <= #1 32'd0;         new_sum <= #1 32'd0;
          new_in  <= #1 32'd0;              rdy     <= #1 1'b0;
          in_n    <= #1 32'd0;         in_n1   <= #1 32'd0;
          acc     <= #1 32'd0;         mux     <= #1 2'd0;
          Ap_n    <= #1 16'd0;         Ai_n    <= #1 16'd0;
          Ad_n    <= #1 16'd0;         set     <= #1 1'b0;
           end
     else
       if(strobe && enable) // get new values
         begin
            in_n1 <= #1 prev_in;
            in_n  <= #1 err;
            new_in<= #1 err;
            acc   <= #1 prev_sum;
            mux   <= #1 2'd0;
            rdy   <= #1 1'b0;
            Ap_n  <= #1 Ap;
            Ai_n  <= #1 Ai;
            Ad_n  <= #1 Ad;
            set   <= #1 1'b1;
         end
       else if (set && !rdy)
         begin
            mux <= #1 mux + 1'b1;
            case(mux)
              2'd1 :
                begin
                   acc <= #1 sum; // result from mux = 0 (Ai*in_n)
                   if(sum[30] && !sum[31]) // clip high
                      new_sum <= #1 32'h40000000;
                   else if(sum[31] && !sum[30]) // clip low
                      new_sum <= #1 32'hc0000000;
                   else
                      new_sum <= #1 sum;// this is sum to carry over
                end
              2'd2 : acc <= #1 sum; // result from mux = 1 (Ad*diff)
              2'd3 :
                begin
                   set <= #1 1'b0;
                   rdy <= #1 1'b1;
                   cont <= #1 sum; // result from mux = 2 (Ap*in_n)
                end
            endcase // case (mux)
         end // if (!rdy)
       else
         rdy <= #1 1'b0;
endmodule // pipelined_pid



//  load gains into l,m,n and Ap, Ai, Ad based on current mux
module rate_gyro_gains (clk, pid_mux, iir_mux,
                        b1, a0, a1, Ap, Ai, Ad,
                                            serial_addr, serial_data, serial_strobe);
   parameter E_AP_ADDR  = 64;
   parameter E_AID_ADDR = 65;
   parameter S_AP_ADDR  = 66;
   parameter S_AID_ADDR = 67;
   parameter Q_AP_ADDR  = 68;
   parameter Q_AID_ADDR = 69;
   parameter L_AP_ADDR  = 70;
   parameter L_AID_ADDR = 71;
   parameter PLL_B_ADDR = 72;
   parameter PLL_A_ADDR = 73;
   parameter PID_B_ADDR = 74;
   parameter PID_A_ADDR = 75;
   parameter resolution = 32;

        input wire clk;
   input wire [1:0] pid_mux, iir_mux;
   output wire [15:0] b1,a0,a1;
   output wire [15:0] Ap,Ai,Ad;

   input wire [6:0]  serial_addr;
   input wire [31:0] serial_data;
   input wire        serial_strobe;

   wire [15:0] E_Ap, E_Ai, E_Ad;
   wire [15:0] S_Ap, S_Ai, S_Ad;
   wire [15:0] Q_Ap, Q_Ai, Q_Ad;
   wire [15:0] L_Ap, L_Ai, L_Ad;
   wire [15:0] pll_a0, pll_a1, pll_b1;
   wire [15:0] pid_a0, pid_a1, pid_b1;

   assign Ap  = pid_mux[1] ? ( pid_mux[0] ? E_Ap : S_Ap )
     : (pid_mux[0] ? Q_Ap : L_Ap);
   assign Ai  = pid_mux[1] ? ( pid_mux[0] ? E_Ai : S_Ai )
     : (pid_mux[0] ? Q_Ai : L_Ai);
   assign Ad  = pid_mux[1] ? ( pid_mux[0] ? E_Ad : S_Ad )
     : (pid_mux[0] ? Q_Ad : L_Ad);
   assign b1 = iir_mux[1] ? ( iir_mux[0] ? pid_b1 : pid_b1 )
     : (iir_mux[0] ? pid_b1 : pll_b1);
   assign a0 = iir_mux[1] ? ( iir_mux[0] ? pid_a0 : pid_a0 )
     : (iir_mux[0] ? pid_a0 : pll_a0);
   assign a1 = iir_mux[1] ? ( iir_mux[0] ? pid_a1 : pid_a1 )
     : (iir_mux[0] ? pid_a1 : pll_a1);

   wire [31:0] E_Aid;
   wire [31:0] Q_Aid;
   wire [31:0] S_Aid;
   wire [31:0] L_Aid;
   wire [31:0] pll_a;
   wire [31:0] pid_a;
   assign E_Ai = E_Aid[15:0];
   assign E_Ad = E_Aid[31:16];
   assign Q_Ai = Q_Aid[15:0];
   assign Q_Ad = Q_Aid[31:16];
   assign S_Ai = S_Aid[15:0];
   assign S_Ad = S_Aid[31:16];
   assign L_Ai = L_Aid[15:0];
   assign L_Ad = L_Aid[31:16];
   assign pll_a0 = pll_a[15:0];
   assign pll_a1 = pll_a[31:16];
   assign pid_a0 = pid_a[15:0];
   assign pid_a1 = pid_a[31:16];

   setting_reg #(E_AP_ADDR) sr_e_ap(.clock(clk),.reset(1'b0),.strobe(serial_strobe),.
       addr(serial_addr),.in(serial_data),.out(E_Ap));
   setting_reg #(E_AID_ADDR) sr_e_aid(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(E_Aid));
   setting_reg #(S_AP_ADDR) sr_s_ap(.clock(clk),.reset(1'b0),.strobe(serial_strobe),.
       addr(serial_addr),.in(serial_data),.out(S_Ap));
   setting_reg #(S_AID_ADDR) sr_s_aid(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(S_Aid));
   setting_reg #(Q_AP_ADDR) sr_q_ap(.clock(clk),.reset(1'b0),.strobe(serial_strobe),.
       addr(serial_addr),.in(serial_data),.out(Q_Ap));
   setting_reg #(Q_AID_ADDR) sr_q_aid(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(Q_Aid));
   setting_reg #(L_AP_ADDR) sr_l_ap(.clock(clk),.reset(1'b0),.strobe(serial_strobe),.
       addr(serial_addr),.in(serial_data),.out(L_Ap));
   setting_reg #(L_AID_ADDR) sr_l_aid(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(L_Aid));
   setting_reg #(PLL_B_ADDR) sr_pll_b(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(pll_b1));
   setting_reg #(PLL_A_ADDR) sr_pll_a(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(pll_a));
   setting_reg #(PID_B_ADDR) sr_pid_b(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(pid_b1));
   setting_reg #(PID_A_ADDR) sr_pid_a(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(pid_a));

endmodule // rate_gyro_gains
```

APPENDIX D

Relay Mode Firmware

```
//  Pipelined gyroscope parameter extractor and controller
module relay_gyro_cont (clk, reset, enable,
                        serial_addr, serial_data, serial_strobe,
                        cx, sx, cy, sy, strobe, phase, L_n, Lcont );
   parameter FREQ_ADDR = 0;
   parameter CONT_ADDR = 0;
   parameter resolution= 32;

   input wire    clk, reset, enable, strobe;
   input wire [15:0] cx;
   input wire [15:0] sx;
   input wire [15:0] cy;
   input wire [15:0] sy;

   input wire [6:0]  serial_addr;
   input wire [31:0] serial_data;
   input wire        serial_strobe;

   output reg  [31:0] phase;

   reg            rdy; // signals finished one sample, wait for another
   reg [1:0]      iir_mux;
   reg            ext_strobe, pid_strobe, iir_strobe;
   wire           ext_rdy, pid_rdy, iir_rdy;
   wire [31:0]    pid_sum, pid_last, pid_cont,
                  pid_new_sum, pid_new_last;
   wire [15:0]    pid_Ap, pid_Ai, pid_Ad;
   wire [15:0]    iir_in, iir_in_hist;
   wire [31:0]    iir_out, iir_out_hist;
   wire [15:0]    iir_b1, iir_a0, iir_a1;

   wire signed [31:0] param; // extracted parameter
   output wire [31:0] L_n; // latched parameter and previous
   output reg  [31:0] Lcont;
   reg [15:0]     pid_err, Llast;
   reg [31:0]     L_iir_out_hist;
   reg [31:0]     Lsum;
   reg [15:0]     Lcont_n0;
   reg [15:0]     Lcont_n1;
   wire [15:0]    Lcont_short;
   wire [31:0]    freq0;
   wire [31:0]   cont_config; // cont_config[3:0] == enable spid,qpid,lpid,epid
   reg [15:0]    cx_n; // latched inputs
   reg [15:0]    sx_n;
   reg [15:0]    cy_n;
   reg [15:0]    sy_n;

   relay_extractor extractor
     (.clk(clk), .reset(reset), .enable(enable), .strobe(ext_strobe),
      .cx(cx_n), .sx(sx_n), .cy(cy_n), .sy(sy_n),
      .out(param), .rdy(ext_rdy));

   pipelined_pid16 pid
     ( .clk(clk), .reset(reset), .enable(enable), .strobe(pid_strobe),
       .err(pid_err), .prev_sum(pid_sum), .prev_in(pid_last),
       .new_in(pid_new_last), .new_sum(pid_new_sum), .cont(pid_cont),
       .rdy(pid_rdy), .Ap(pid_Ap), .Ai(pid_Ai), .Ad(pid_Ad) );

   pipelined_iir1 iir
     ( .clk(clk), .reset(reset), .enable(enable), .strobe(iir_strobe),
       .in(iir_in), .out(iir_out), .rdy(iir_rdy),
       .in_hist(iir_in_hist),.out_hist(iir_out_hist),
       .b1(iir_b1),.a0(iir_a0),.a1(iir_a1) );

   relay_gyro_gains rgg
     ( .clk(clk),
       .b1(iir_b1), .a0(iir_a0), .a1(iir_a1),
       .Ap(pid_Ap), .Ai(pid_Ai), .Ad(pid_Ad),
       .serial_addr(serial_addr), .serial_data(serial_data),
       .serial_strobe(serial_strobe) );

   assign L_n = pid_err;
   assign pid_sum  = Lsum;
   assign pid_last = Llast;
   assign iir_in   = Lcont_n0;
   assign iir_in_hist  = Lcont_n1;
   assign iir_out_hist = L_iir_out_hist;

   setting_reg #(FREQ_ADDR) sr_freq0(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(freq0));
   setting_reg #(CONT_ADDR) sr_config(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(cont_config));

   assign Lcont_short = Lcont[31:16];

   always @(posedge clk)
     if (reset)
       phase <= #1 0;
     else if (enable & strobe)
       phase <= #1 phase + freq0 + Lcont;

   always @(posedge clk)
      if (reset)
         begin
            rdy <= #1 1'b1;                    pid_strobe <= #1 1'd0;
            iir_strobe <= #1 1'd0;             pid_err <= #1 16'd0;
            L_iir_out_hist <= #1 32'd0;        Lcont_n0  <= #1 16'd0;
            Lcont_n1  <= #1 16'd0;             Lsum <= #1 32'd0;
            Llast <= #1 16'd0;                 Lcont <= #1 32'd0;
            cx_n <= #1 16'd0;                  sx_n <= #1 16'd0;
            cy_n <= #1 16'd0;                  sy_n <= #1 16'd0;
         end
      else if (enable & strobe)
         begin
            rdy <= #1 1'b0;
            cx_n <= #1 cx; // sample inputs
            sx_n <= #1 sx; // sample inputs
            cy_n <= #1 cy; // sample inputs
            sy_n <= #1 sy; // sample inputs
```

```
              iir_mux <= #1 2'h0;
              // stobe kickstarts new extract sequence, rest are driven by rdy
              ext_strobe <= #1 1'b1;
           end
        else if (!rdy && enable && (ext_rdy | iir_rdy | pid_rdy))
           begin
              if(ext_rdy)
                 begin
                    pid_strobe <= #1 1'b1;
                    pid_err <= #1 (param[31] & (param < 32'hff800000)) ? 16'h8000 : (~
                        param[31] & (param > 32'h007fffff) ? 32'h7fff : param[23:8]+
                        param[7]);
                 end
              else
                 ext_strobe <= #1 1'b0;
           if(pid_rdy) // latch history of PID
              begin
                 pid_strobe <= #1 1'b0; // wait for new data
                 iir_strobe <= #1 1'b1;
                 Lcont_n0 <= #1 pid_cont[31:16]+pid_cont[15];
                 Lcont_n1 <= #1 Lcont_n0;
                 Lsum  <= #1 cont_config[3] ? pid_new_sum : 32'b0;
                 Llast <= #1 pid_new_last;
              end // if (pid_rdy)
           if(iir_rdy) // latch history of IIR
             begin
                iir_mux <= #1 iir_mux + 1'b1;
                case (iir_mux)
                  2'd0:
                    begin
                       Lcont <= #1 iir_out;
                       L_iir_out_hist <= #1 Lcont;
                       rdy <= #1 1'b1;
                                                         iir_strobe <= #1 1'b0;
                    end
                endcase
             end // if (iir_rdy)
         end // if (!rdy && enable && (ext_rdy | iir_rdy | pid_rdy))
       else // turn off strobes
         begin
            ext_strobe <= #1 1'b0;
            iir_strobe <= #1 1'b0;
            pid_strobe <= #1 1'b0;
         end

endmodule // relay_gyro_cont

// this only extracts Li, much simpler than WA
// the implementation could be simpler, but this should be more
// easily expanded to the full WA control
module relay_extractor (clk, reset, enable, strobe,
                        cx, sx, cy, sy, out, rdy);
   parameter RES       = 32;

   input wire clk, reset, enable, strobe;
   output reg rdy;
   input wire [15:0]  cx, sx, cy, sy;
   reg signed [15:0]  cx_l, sx_l, cy_l, sy_l;
   wire signed [30:0] csx, csy; // intermediate calculations
   output reg [31:0]  out;

   wire signed [15:0]        cM; // muxed c input
   wire signed [15:0]        sM; // muxed s input
   wire signed [31:0]        sum;
   wire signed [30:0]        prod;
   wire signed [31:0]        prod_e;
   reg signed [31:0]         acc;
   reg [1:0]           mux;
   reg                 set;

   assign prod_e = {prod[30],prod};
   assign sum    = acc + prod_e;

   mult mult(.clock(clk),.x(cM),.y(sM),
             .product(prod),.enable_in(1'b1),
             .enable_out());

   assign cM = mux[0] ? cy_l : cx_l;
   assign sM = mux[0] ? sy_l : sx_l;

   always @(posedge clk)
         begin
        if(reset)
          begin
             out  <= #1 32'b0;                rdy  <= #1 1'b0;
             cx_l <= #1 16'b0;            sx_l <= #1 16'b0;
             cy_l <= #1 16'b0;            sy_l <= #1 16'b0;
             acc  <= #1 32'd0;            mux  <= #1 2'd0;
             set  <= #1 1'b0;
                  end
        else
          begin
             if(strobe & enable)
               begin
                  cx_l <= #1 cx;
                  sx_l <= #1 sx;
                  cy_l <= #1 cy;
                  sy_l <= #1 sy;
                  acc  <= #1 32'd0;
                  rdy  <= #1 1'b0;
                  mux  <= #1 1'b0;
                  set  <= #1 1'b1;
               end
             else if (set && !rdy)
               begin
                  mux <= #1 mux + 2'b1;
                  case(mux)
                    2'd1 : acc <= #1 sum; // acc += cx*sx
                    2'd2 :
                      begin
                         out <= #1 sum;  // acc + cy*sy
                         set <= #1 1'b0; // disable mux (set by strobe)
                         rdy <= #1 1'b1; // strobe ready to parent module
                      end
                  endcase // case (mux)
               end // else: !if((strobe | rdy) & enable)
             else
               rdy <= #1 1'b0;
          end // else: !if(reset)
      end // always @ (posedge clk)
endmodule // relay_extractor

// Implement PID that rounds to 16 bits
module pipelined_pid16 (clk, reset, enable, strobe,
                        err, prev_sum, prev_in, new_in, new_sum, cont, rdy,
                        Ap, Ai, Ad);
   parameter RES       = 32;
   parameter SCALE     = 5;
   parameter ISCALE    = 0;
   parameter DSCALE    = 0;
   input wire clk;
   input wire reset;
   input wire enable;
   input wire strobe;

   input wire [15:0] err; // this is the input to control
   input wire [RES-1:0] prev_sum;
   input wire [15:0] prev_in;
   output reg [RES-1:0] cont;
   output reg [RES-1:0] new_sum;
   output reg [15:0] new_in;
   output reg            rdy;
   input wire [15:0]     Ap,Ai,Ad;

   wire [15:0]  gain;  // muxed gain
   wire [15:0]  val;   // muxed memory
   wire [31:0]  sum;
   wire [31:0]  diff;
   wire [30:0]  prod;
   wire [31:0]  prod_e;
   reg [31:0]   acc;
   reg [2:0]    mux;
   reg [15:0]   Ap_n,Ai_n,Ad_n;
   reg signed [15:0] in_n;
   reg signed [15:0] in_n1;
   reg               set;

   mult mult(.clock(clk),.x(val),.y(gain),
            .product(prod),.enable_in(1'b1),
            .enable_out());
   // y[n] = pg*x[n] + sum
   // sum  = sum[n-1] + pi*x[n] + pd*(x[n]-x[n-1])
   assign gain = mux[1] ? Ap_n : (mux[0] ? Ad_n : Ai_n);
   // when mux == 0, in_n is previous input,
   // when mux == 1, in_n will be the newly sampled input
   assign diff  = in_n - in_n1;
   assign val   = mux[1] ? in_n : (mux[0] ? diff : in_n);
   // scaling for sample rat, mux is +1 from gain and val. Use same scale for P and D
        gains
   assign prod_e = {prod[30],prod[30:0]};

   assign sum = acc + prod_e;

   always @(posedge clk)
     if(reset)
       begin
          cont    <= #1 32'd0;           new_sum <= #1 32'd0;
          new_in  <= #1 32'd0;           rdy     <= #1 1'b0;
          in_n    <= #1 16'd0;           in_n1   <= #1 16'd0;
          acc     <= #1 32'd0;           mux     <= #1 2'd0;
          Ap_n    <= #1 16'd0;           Ai_n    <= #1 16'd0;
          Ad_n    <= #1 16'd0;           set     <= #1 1'b0;
       end
     else
       if(strobe && enable) // get new values
         begin
            in_n1 <= #1 prev_in;
            in_n  <= #1 err;
            new_in<= #1 err;
            acc   <= #1 prev_sum;
            mux   <= #1 2'd0;
            rdy   <= #1 1'b0;
            Ap_n  <= #1 Ap;
            Ai_n  <= #1 Ai;
            Ad_n  <= #1 Ad;
            set   <= #1 1'b1;
         end
       else if (set && !rdy)
         begin
            mux <= #1 mux + 1'b1;
            case(mux)
              2'd1 :
                begin
                   acc <= #1 sum; // result from mux = 0 (Ai*in_n)
```

```
                    if(sum[30] && !sum[31]) // clip high
                       new_sum <= #1 32'h40000000;
                    else if(sum[31] && !sum[30]) // clip low
                       new_sum <= #1 32'hc0000000;
                    else
                       new_sum <= #1 sum;// this is sum to carry over
                 end
               2'd2 : acc <= #1 sum; // result from mux = 1 (Ad*diff)
               2'd3 :
                 begin
                    set <= #1 1'b0;
                    rdy <= #1 1'b1;
                    cont <= #1 sum; // result from mux = 2 (Ap*in_n)
                 end
             endcase // case (mux)
          end // if (!rdy)
        else
          rdy <= #1 1'b0;
endmodule // pipelined_pid16


//  load gains into l,m,n and Ap, Ai, Ad based on current mux
module relay_gyro_gains (clk,
                        b1, a0, a1, Ap, Ai, Ad,
                                         serial_addr, serial_data, serial_strobe);
   parameter L_AP_ADDR  = 70;
   parameter L_AID_ADDR = 71;
   parameter PLL_B_ADDR = 72;
   parameter PLL_A_ADDR = 73;
   parameter resolution = 32;

   input wire clk;
   output wire [15:0] b1,a0,a1;
   output wire [15:0] Ap,Ai,Ad;

   input wire [6:0]  serial_addr;
   input wire [31:0] serial_data;
   input wire        serial_strobe;

   wire [15:0] L_Ap, L_Ai, L_Ad;
   wire [15:0] pll_a0, pll_a1, pll_b1;

   assign Ap = L_Ap;
   assign Ai = L_Ai;
   assign Ad = L_Ad;
   assign b1 = pll_b1;
   assign a0 = pll_a0;
   assign a1 = pll_a1;

   setting_reg #(L_AP_ADDR)  sr_l_ap(.clock(clk), .reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(L_Ap));
   setting_reg #(L_AID_ADDR) sr_l_aid(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out({L_Ad,L_Ai}));
   setting_reg #(PLL_B_ADDR) sr_pll_b(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out(pll_b1));
   setting_reg #(PLL_A_ADDR) sr_pll_a(.clock(clk),.reset(1'b0),.strobe(serial_strobe)
       ,.addr(serial_addr),.in(serial_data),.out({pll_a1,pll_a0}));

endmodule // gyro_gains
```

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] M. Perlmutter and L. Robin, "High-Performance, low cost inertial MEMS: a market in motion!" in *IEEE/ION PLANS 2012*. Myrtle Beack, CA: IEEE, Apr. 2012.

[2] S. Saraswathy, J. Geen, and J. Chang, "High performance gyro with fast startup time, high range, wide bandwidth, low noise and excellent vibration immunity," Myrtle Beack, SC, Apr. 2012.

[3] J. Cho, "Environmentally resistant rate and rate-integrating gyroscopes," Ph.D. dissertation, Dept. Elect. Eng., Univ. Michigan, Ann Arbor, 2012.

[4] D. Meyer and D. Rozelle, "milli-HRG inertial navigation system," Myrtle Beack, SC, Apr. 2012.

[5] C. Painter and A. Shkel, "Experimental evaluation of a control system for an absolute angle measuring micromachined gyroscope," in *Sensors, 2005 IEEE*, Nov. 2005, pp. 1084–1087.

[6] I. P. Prikhodko, S. A. Zotov, A. A. Trusov, and A. M. Shkel, "Foucault pendulum on a chip: Rate integrating silicon MEMS gyroscope," *Sensors and Actuators A: Physical*, vol. 177, pp. 67–78, Apr. 2012.

[7] D. Keymeulen, C. Peay, D. Foor, T. Trung, A. Bakhshi, P. Withington, K. Yee, and R. Terrile, "Control of MEMS disc resonance gyroscope (DRG) using a FPGA platform," in *Aerospace Conference, 2008 IEEE*, Mar. 2008, pp. 1 –8.

[8] A. A. Trusov, I. Chepurko, A. R. Schofield, and A. M. Shkel, "A standalone programmable signal processing unit for versatile characterization of MEMS gyroscopes," in *2007 IEEE Sensors*. IEEE, Oct. 2007, pp. 244–247.

[9] S. Park, R. Horowitz, and C. Tan, "Dynamics and control of a MEMS angle measuring gyroscope," *Sensors and Actuators A: Physical*, vol. 144, no. 1, pp. 56–63, May 2008.

[10] J. A. Geen, S. J. Sherman, J. F. Chang, and S. R. Lewis, "Single-chip surface micromachined integrated gyroscope with 50/h allan deviation," *IEEE Journal of Solid-State Circuits*, vol. 37, no. 12, pp. 1860– 1866, Dec. 2002.

[11] S. Rombach, T. Northemann, M. Maurer, M. Dienger, and Y. Manoli, "Modulated electro-mechanical continuous-time lowpass sigma-delta-modulator for micromachined gyroscopes." IEEE, Jun. 2011, pp. 1092–1095.

[12] M. Ferguson, D. Keymeulen, K. Hayworth, B. Blaes, C. Peay, K. Yee, E. MacDonald, and D. Foor, "A hardware platform for tuning of MEMS devices using Closed-Loop frequency response," in *Aerospace Conference, 2005 IEEE*, Mar. 2005, pp. 1 –7.

[13] E. J. Loper and D. D. Lynch, "Sonic vibrating bell gyro," Jun. 1979, U.S. Classification: 73/504.13 International Classification: : G01P 904; G01C 1956.

[14] M. W. Putty, "A micromachined vibrating ring gyroscope." 1992.

[15] B. Friedland and M. Hutton, "Theory and error analysis of vibrating-member gyroscope," *IEEE Transactions on Automatic Control*, vol. 23, no. 4, pp. 545–556, Aug. 1978.

[16] V. Zhuravlev, "Oscillation shape control in resonant systems," *Journal of Applied Mathematics and Mechanics*, vol. 56, no. 5, pp. 725–735, 1992.

[17] D. D. Lynch, A. Mathews, and G. T. Varty, "Innovative mechanizations to optimize inertial sensors for high or low rate operations," Stuttgart, Germany, Sep. 1997.

[18] S. Smyser, "Si-Ware systems," http://www.si-ware.com/ShowPage.aspx?PID=144, May 2012. [Online]. Available: http://www.si-ware.com/ShowPage.aspx?PID=144

[19] D. D. Lynch, "Vibratory gyro analysis by the method of averaging," in *2nd St. Petersburg Int. Conf. on Gyroscopic Technology and Navigation*, St. Petersburg, Russia, 1995.

[20] D. Piyabongkarn, R. Rajamani, and M. Greminger, "The development of a MEMS gyroscope for absolute angle measurement," *IEEE Transactions on Control Systems Technology*, vol. 13, no. 2, pp. 185–195, Mar. 2005.

[21] A. Sharma, M. F. Zaman, and F. Ayazi, "A smart angular rate sensor system," in *IEEE Sensors 2007*, Oct. 2007, pp. 1116–1119.

[22] D. Schwartz, D. J. Kim, and R. T. M'Closkey, "Frequency tuning of a disk resonator gyro via mass matrix perturbation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 131, no. 6, p. 061004, 2009.

[23] Dong Joon Kim and R. M'Closkey, "A systematic method for tuning the dynamics of electrostatically actuated vibratory gyros," *IEEE Transactions on Control Systems Technology*, vol. 14, no. 1, pp. 69–81, Jan. 2006.

[24] M. W. Putty, “Micromachined vibrating ring gyroscope.” Ph.D. dissertation, Dept. Elect. Eng., Univ. Michigan, Ann Arbor, 1995.

[25] B. J. Gallacher, J. Hedley, J. S. Burdess, A. J. Harris, A. Rickard, and D. O. King, “Electrostatic correction of structural imperfections present in a microring gyroscope,” *Journal of Microelectromechanical Systems*, vol. 14, no. 2, pp. 221–234, Apr. 2005.

[26] Sangkyung Sung, Woon-Tahk Sung, Changjoo Kim, Sukchang Yun, and Young Jae Lee, “On the Mode-Matched control of MEMS vibratory gyroscope via Phase-Domain analysis and design,” *IEEE/ASME Transactions on Mechatronics*, vol. 14, no. 4, pp. 446–455, Aug. 2009.

[27] W. A. Clark and J. A. Geen, “Offset detection and compensation for micromachined inertial sensors,” publication number: US 2011/0041609 A1 U.S. Classification: 73/514.29.

[28] G. Casinovi, W. Sung, M. Dalal, A. Shirazi, and F. Ayazi, “Electrostatic self-calibration of vibratory gyroscopes,” in *Micro Electro Mechanical Systems (MEMS), 2012 IEEE 25th International Conference on*, Feb. 2012, pp. 559 – 562.

[29] R. P. Feynman, *The Feynman lectures on physics*, Jan. 1989, vol. 1.

[30] W. Sung, M. Dalal, and F. Ayazi, “A mode-matched 0.9 MHZ single proof-mass dual-axis gyroscope,” in *Solid-State Sensors, Actuators and Microsystems Conference (TRANSDUCERS), 2011 16th International*, Jun. 2011, pp. 2821 –2824.

[31] J. Wu, G. Fedder, and L. Carley, “A low-noise low-offset capacitive sensing amplifier for a 50- mu;g/ radic;Hz monolithic CMOS MEMS accelerometer,” *Solid-State Circuits, IEEE Journal of*, vol. 39, no. 5, pp. 722 – 730, May 2004.

[32] B. Gallacher, J. Burdess, and K. Harish, “A control scheme for a mems electrostatic resonant gyroscope excited using combined parametric excitation and harmonic forcing,” *Journal of Micromechanics and Microengineering*, vol. 16, no. 2, pp. 320 – 331, 2006, excitation scheme;Oscillator scheme;Parametric excitation;Rate resolution;.

[33] Ettus Research, “USRP1,” Mar. 2012. [Online]. Available: https://www.ettus.com/product/details/USRP-PKG

[34] J. Cho, J. Gregory, and K. Najafi, “Single-crystal-silicon vibratory cylinderical rate integrating gyroscope (CING),” in *Solid-State Sensors, Actuators and Microsystems Conference (TRANSDUCERS), 2011 16th International*, Jun. 2011, pp. 2813–2816.

[35] ——, "High-Q, 3kHz Single-Crystal-Silicon cylindrical Rate-Integrating gyro (CING)," in *Micro Electro Mechanical Systems (MEMS), 2012 IEEE 25th International Conference on*, Paris, France, Feb. 2012, pp. 172–175.

[36] IEEE, "IEEE standard specification format guide and test procedure for Single-Axis laser gyros," IEEE Aerospace and Electronic Systems Society, Standard 647-2006, 2006.

[37] SciPy, "SciPy," Mar. 2012. [Online]. Available: http://www.scipy.org/

[38] Y. Stebler, S. Guerrier, J. Skaloud, and M. Victoria-Feser, "A framework for inertial sensor calibration using complex stochastic error models," Myrtle Beack, SC, Apr. 2012.

[39] R. J. Vaccaro and A. S. Zaki, "Statistical modeling of rate gyros," *IEEE Transactions on Instrumentation and Measurement*, vol. 61, no. 3, pp. 673–684, Mar. 2012.

[40] Z. Berman, "Inertial sensors: Further developments in Low-Cost calibration and testing," Myrtle Beack, SC, Apr. 2012.

[41] GNU Radio, "GNU radio," Mar. 2012. [Online]. Available: http://www.gnuradio.org

[42] Agilent, "E5061B-005 LF-RF network analyzer option + 005 impedance analysis option," Mar. 2012. [Online]. Available: http://www.home.agilent.com/agilent/product.jspx?pid=1944859

[43] Tektronix, "RSA5000," Mar. 2012. [Online]. Available: http://www.tek.com/spectrum-analyzer/rsa5000

[44] R. Stewart, "An overview of sigma delta ADCs and DAC devices," in *Oversampling and Sigma-Delta Strategies for DSP, IEE Colloquium on*, Nov. 1995, pp. 1/1 –1/9.

[45] T. Gabrielson, "Mechanical-thermal noise in micromachined acoustic and vibration sensors," *Electron Devices, IEEE Transactions on*, vol. 40, no. 5, pp. 903 –909, May 1993.

[46] R. Leland, "Mechanical-thermal noise in mems gyroscopes," *IEEE Sensors Journal*, vol. 5, no. 3, pp. 493 – 500, 2005/06/, mEMS gyroscopes;mechanical thermal noise;vibrational gyroscope;random walk;noise equivalent rotation rate;noise spectral density;noise component;synchronous demodulation;.

[47] M. Kirkko-Jaakkola, J. Collin, and J. Takala, "Bias prediction for MEMS gyroscopes," *Sensors Journal, IEEE*, vol. 12, no. 6, pp. 2157 –2163, Jun. 2012.

[48] T. Lee and A. Hajimiri, "Oscillator phase noise: a tutorial," *Solid-State Circuits, IEEE Journal of*, vol. 35, no. 3, pp. 326 –336, Mar. 2000.

[49] H. Johari and F. Ayazi, "High-frequency capacitive disk gyroscopes in (100) and (111) silicon," in *2007 IEEE 20th International Conference on Micro Electro Mechanical Systems (MEMS)*, jan. 2007, pp. 47 –50.

[50] J. M. Muller, *Elementary Functions Algorithms and Implementation.* Boston, MA: Birkhuser Boston, 2006.

[51] A. Matthews, G. T. Varty, C. Li, and D. D. Lynch, "United states patent: 5801310 - vibratory rotation sensor with whole-angle tracking," Sep. 1998.

[52] I. P. Prikhodko, S. A. Zotov, A. A. Trusov, and A. M. Shkel, "Foucault pendulum on a chip: angle measuring silicon MEMS gyroscope," in *2011 IEEE 24th International Conference on Micro Electro Mechanical Systems (MEMS).* IEEE, Jan. 2011, pp. 161–164.

[53] V. Zhuravlev, "Theoretical foundations of solid-state wave gyroscopes," *Mechanics of Solids*, vol. 28, no. 3, pp. 3–15, 1993.

[54] J. Gregory, J. Cho, and K. Najafi, "MEMS rate and rate-integrating gyroscope control with commercial software defined radio hardware," in *Solid-State Sensors, Actuators and Microsystems Conference (TRANSDUCERS), 2011 16th International*, Jun. 2011, pp. 2394–2397.

[55] ——, "Characterization and control of a high-Q MEMS inertial sensor using low-cost hardware," in *IEEE/ION PLANS 2012.* Myrtle Beack, CA: IEEE, Apr. 2012.

[56] Free Software Foundation, "GSL - GNU scientific library - GNU project," Mar. 2012. [Online]. Available: http://www.gnu.org/software/gsl/

[57] D. Schwartz and R. T. M'Closkey, "Decoupling of a disk resonator from linear acceleration via mass matrix perturbation," *Journal of Dynamic Systems, Measurement, and Control*, vol. 134, no. 2, p. 021005, 2012.

[58] Y. K. Zhbanov, "Amplitude control contour in a hemispherical resonator gyro with automatic compensation for difference in q -factors," *Mechanics of Solids*, vol. 43, no. 3, pp. 328–332, Jun. 2008.