Embedded Controls Final Project

Lucas Uecker

12-5-10

Abstract

This paper outlines a project to correct a gyrocompass against both mechanical noise and the processive force the gyroscope produces. A gain-scheduled proportional control was used to maintain a specific angle. The angle of the gyroscope was measured using a potentiometer and a MSP430 microcontroller was used for active control.

Introduction

Gyrocompasses are a effective and accurate method for determining change in orientation with respect to a reference angle. This occurs to to an effect called procession, which is a force that occurs when a torque vector is perturbed. The torque vector is defined as the the moment of inertia multiplied by the angular velocity. The direction of the vector is the right-hand-normal direction to the plane of rotation.

$$\tau = I\omega \quad (1)$$

The procession vector is then

$$dL = -d\tau \quad (2)$$

or

$$dL/dt = -d\tau/dt \quad (3)$$

For example, if one end of a gyroscope were supported and the other allowed to fall, the processive force L would give equal and opposite force to a gravitational force g:

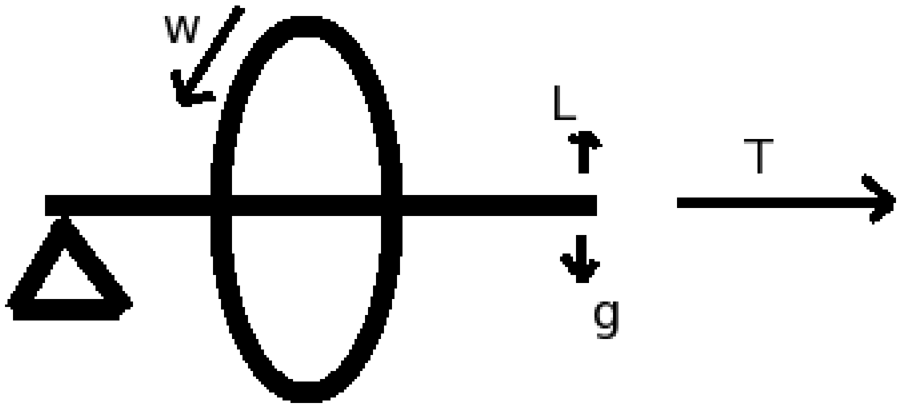

Figure 1: The processive force is enough to counter gravitational acceleration

If a device such as a flywheel with constant rotation is allowed to move freely, it will maintain the same angle with respect to it's axis. This acts as a form of compass, with the axis of the gyroscope pointing in a constant direction with respect to a global frame of reference, despite angular perturbations in the local frame.

Background

This project involves a flywheel with a motor to maintain it's energy storage and allowed to rotate horizontally. The angle of the gyroscope is measured using a potentiometer. The voltage on the potentiometer is measured by the ADC module of a Texas Instruments MSP4302231 microcontroller. The MSP430 calculates an error based on difference between the measured value and the value desired and applies PWM and direction data to a motor controller to correct angle.

Compared to a generic control system, the voltage of the potentiometer is the feedback term, the PWM and direction bits are the control signal and the processive force as well as vibration and system noise are disturbance signals.

The specific control system is a gain-scheduled proportional control. A PD or PID control system is not used due to unreliable behavior during testing. The control motor is linked to the gyroscope via a rubber band and pulley system that does not function in a manner consistent enough to provide accurate data to a PD system. Instead, the controller uses proportional control, where the control variable is directly proportional to the error:

$$C(t)=a*E(t) \quad (4)$$

Normally, proportional control is not used in practice due to the fact that it generally leads to overshoot and is only guaranteed mathematically to asymptotically approach a desired variable. In this system, however, a small range of variables is used instead of a specific variable. Accuracy is slightly diminished in this scheme, but the range comprises a range of error of about 5 degrees, which is taken as acceptable. There is also enough friction in the system to reduce the problem of overshoot.

Gain scheduling is a method of changing the control system used depending on the situation the control system or plant is in. This is used to take advantage of locally linear behavior of control systems instead of finding a single control system that is globally effective. In this system, if the error is sufficient, a constant control signal is applied. Once the error is reduced, the system switches to using the proportional controller.

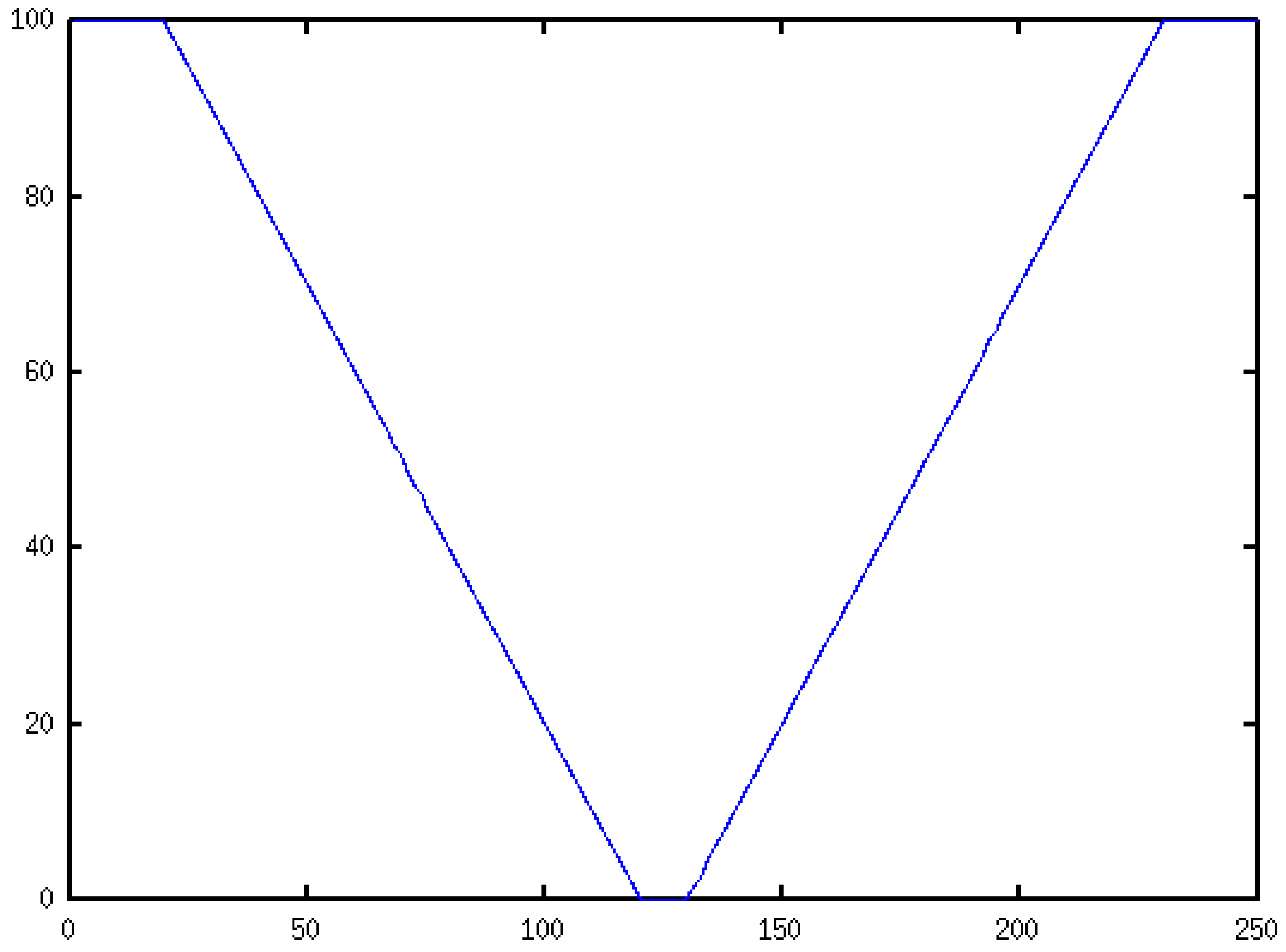

Figure 1: Error vs. Relative PWM

Figure 1 shows the general scheme used. Beyond a certain error in either direction (20 or 230 in the graph), the controller give a constant large correction. In the center, a range 10 values is given no control signal to correct. Between the constant correction and no correction, the linear proportional control is used.

Note Figure 1 is relative PWM duty cycle. In the physical system, 50% is approximately the minimum duty cycle to have reliable movement from the control motor. The minimum duty cycle used was approximately 66%. It was also decided to avoid 100% duty cycle to prevent overheating the motor controller, so the maximum duty cycle used was approximately 90%. These constraints had the fortunate effect of causing the ratio of error to duty cycle to be approximately 1. This allowed the proportional control to be directly determined by the error and the processor spared from some calculation.

Results

Oscilloscope traces of the system:

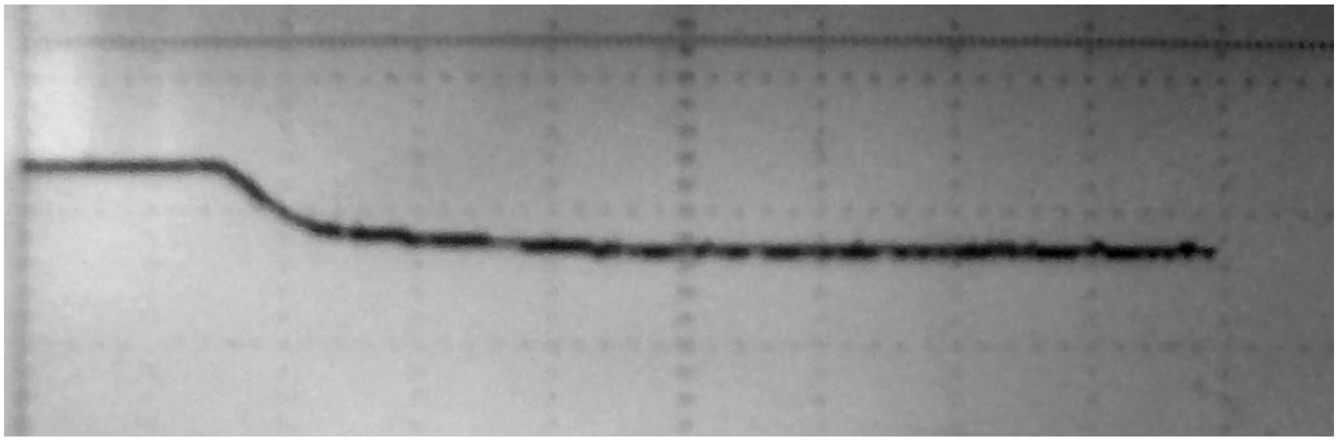
Figure 2: The system correcting a 90 degree error (fast time scale)

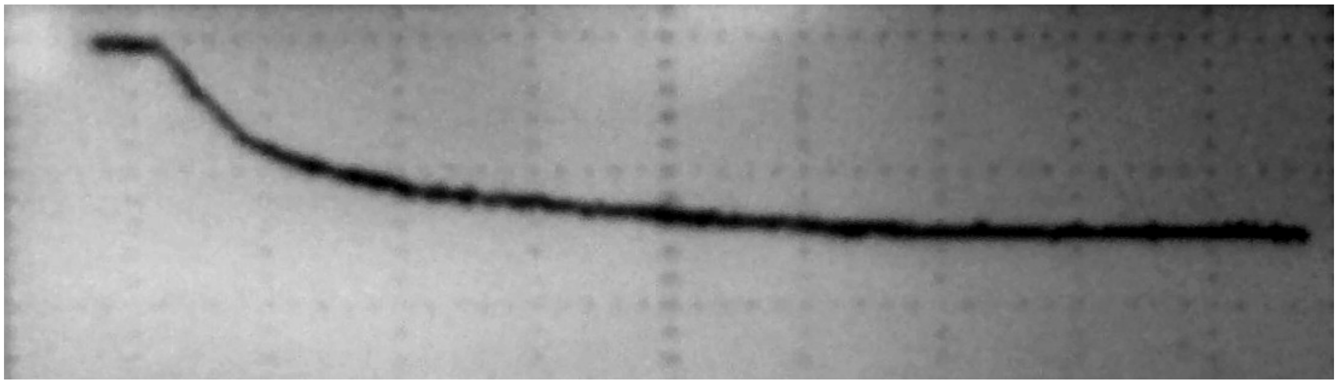
Figure 3: Another 90 degree correction (fast time scale)

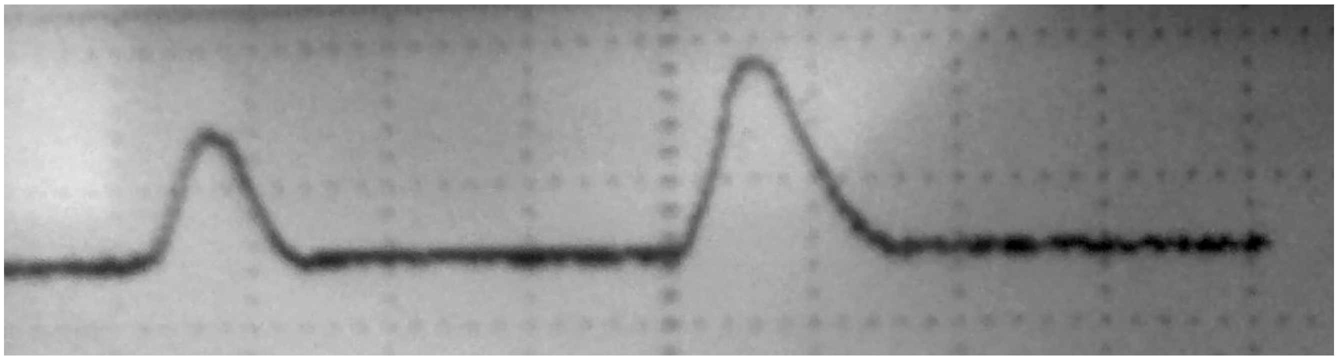
Figure 4: Manual Perturbations of the system (slower time scale)

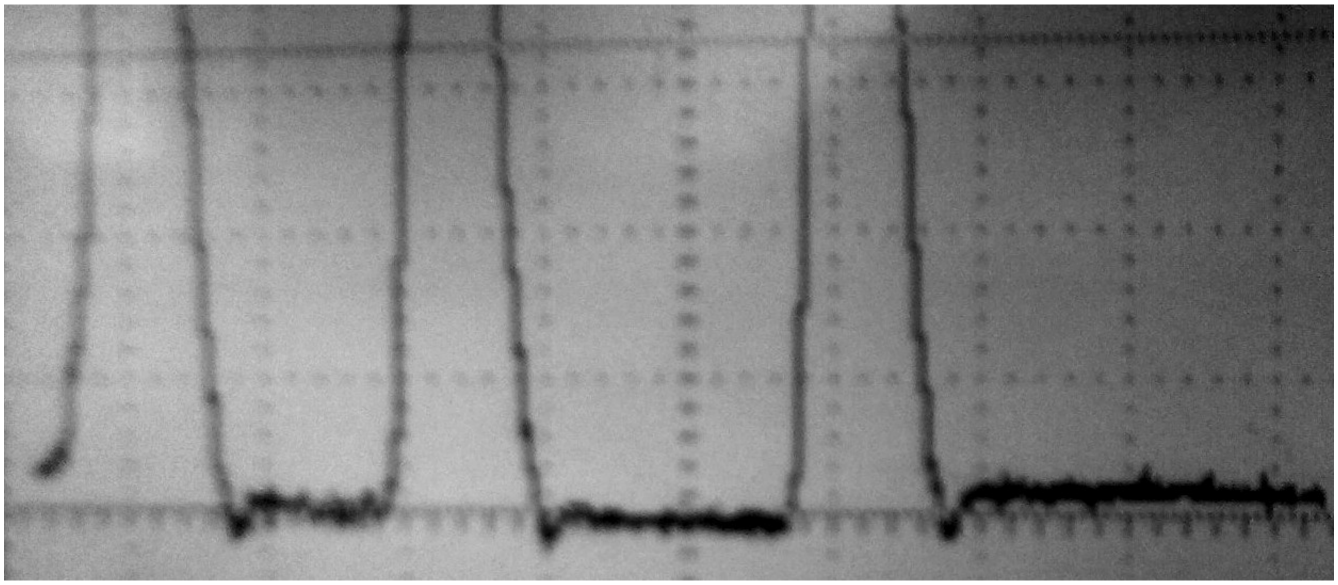
Figure 5: Closer look at Perturbation Traces

The system corrects the 90 degree error in approximately 500ms with approximately 3% overshoot.

Conclusion

The system generally performed as designed. Adequate control was achieved, though friction and other non-linearity did degrade performance. The control motor also seems to be asymmetric in performance, causing the system to work better in one direction than the other.

Appendix A- Program code

```
#include <io.h>          //header files
#include <signal.h>

#define LED0 0x01    //global constants
#define LED1 0x40
#define BUTTON 0x08
#define TXD 0x02
#define RXD 0x04

static void __inline__ delay(register unsigned int n);

void main(void){
WDTCTL = WDTPW + WDTHOLD;                // Stop WDT
ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
ADC10CTL1 = INCH_3;                    // input the ADC at P1.3
ADC10AE0 |= BUTTON;                      // PA.3 ADC option select

P1DIR |= (LED1 + LED0 + TXD + RXD);                          // Set P1.0-2 to output direction
P1SEL |= LED1;                                       // P1.3 to TA0.1
P1OUT |= (LED0 + TXD + RXD);                         //assigning pins to port 1
CCR0 = 1000-1;                          // PWM Period
CCTL1 = OUTMOD_7;                       // CCR1 reset/set
TACTL = TASSEL_2 + MC_1;                // SMCLK, up mode

P1OUT |= 0x05;                          //initializing output to binary 101

while(1){
        ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
        __bis_SR_register(CPUOFF + GIE);       // LPM0, ADC10_ISR will force exit

        if (ADC10MEM < 0x1F0){
                if(ADC10MEM > 0x100){

                        CCR1 = (0x1EE - ADC10MEM) + 0x29A;           //proportional control
                        P1OUT = ~0x05;                  // Toggle control bits
                }else{
                        CCR1 = 900;                             //constant control
                        P1OUT = ~0x05;
                }
        }if(ADC10MEM > 0x1FF){
                if(ADC10MEM < 0x2EF){

                        CCR1 = (ADC10MEM - 0x1FF) + 0x29A;           //proportional control
                        P1OUT = 0x05;                   // Set control bits
                }else{
                        CCR1 = 900;                     //constant control
                        P1OUT = 0x05;
                }
        }else{
                CCR1 = 100;                             // dead zone
        }
        delay(10);                              //delay
}
}

// ADC10 interrupt service routine
//#pragma vector=ADC10_VECTOR
interrupt(ADC10_VECTOR) ADC10_ISR(void){
        __bic_SR_register_on_exit(CPUOFF);     // Clear CPUOFF bit from 0(SR)
}

// Delay Routine from mspgcc help file
static void __inline__ delay(register unsigned int n){
        __asm__ __volatile__ (
        "1: \n"
        " dec %[n] \n"
        " jne 1b \n"
        : [n] "+r"(n));
}
```