Term Project 2: Electronic Directional Gyro MAE 5483

Charles O'Neill

10 December 2004

1 Introduction

This project developed an one degree-of-freedom electronic aircraft directional gyro (DG). The mechanical inertial gyro in a 1960's vintage DG was removed and replaced with an stepping motor driven by a micro controller connected to an electronic rate gyro.

2 Theory

The governing equation is the aircraft strap-down equation for rigid body kinematics. Using Euler angles (Φ Θ Ψ), the strap-down equation is:

$$\begin{bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\Phi\tan\theta & \cos\Phi\tan\theta \\ 0 & \cos\Phi & -\sin\Phi \\ 0 & \sin\Phi\sec\theta & \cos\Phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

where Φ, Θ, and Ψ are the roll, pitch, and yaw angles in the inertial frame, and p, q, and r are the body fixed rates. Assuming one degree-of-freedom (yaw) with zero roll and pitch angles, the matrix equation reduces to a single first order ODE:

$$\dot{\Psi}(t) = r(t) \quad \text{with} \quad \Psi(0) = \Psi_0$$

The numerical discrete-time ODE solution will assume a zero order hold on rate:

$$\Psi(k+1) = \Psi(k) + r(k)\Delta T$$

3 Hardware

This section discusses the mechanical and electrical hardware required. An overall electrical schematic is shown in Figure 1. The hardware consists of several major parts: the stepping motor, Driver, Directional Gyro, Rate Gyro, and the PIC micro controller. These major parts are discussed in more detail below.

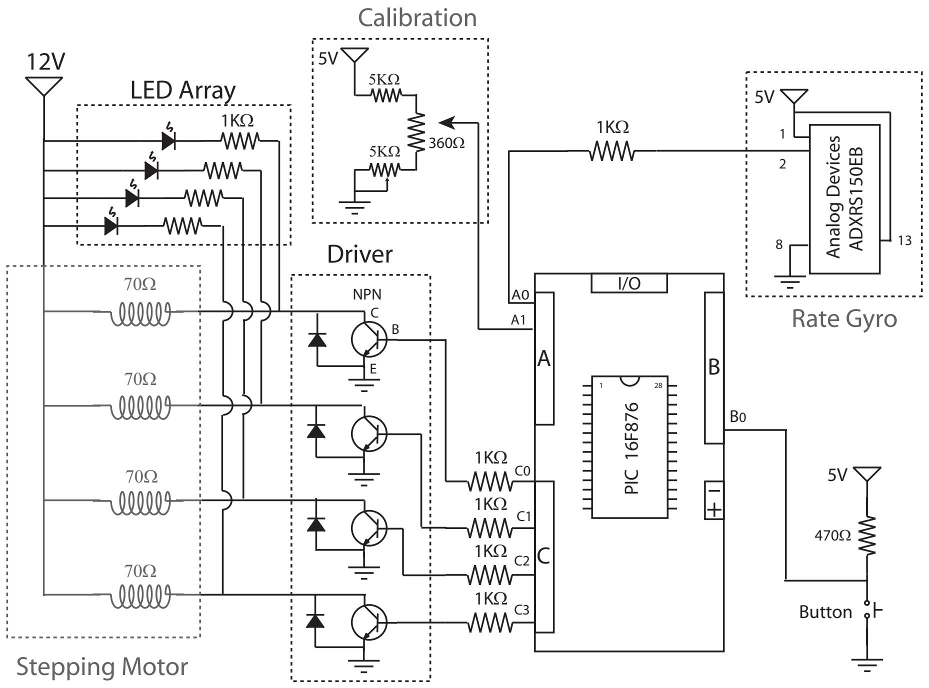

Figure 1: Schematic

3.1 Stepping Motor

This project requires precise mechanical angular motions. Stepping motors are a class of multipole-multiphase DC motors with precision tracking capabilities. The stepping motor rotates in finite *steps* by energizing sequential motor phases; any further rotation requires energizing the next phase. Clockwise rotation require energizing —in sequence— the phases: 1, 2, 3, 4, and so on. Counterclockwise rotation requires the reverse order: 4, 3, 2, 1. Energizing each phase requires a voltage step to the appropriate coil(s) provided by a driver. By necessity, multiple step sequences are required for one revolution. Common stepping motor resolutions are 200 and 400 steps —1.8° and 0.9° respectively. Half stepping recognizes that two adjacent phases can be energized simultaneously to move the rotor between steps; the price of half-stepping is double the current draw with 2 active phases rather than 1.

This project's motor (Fig. 2) was salvaged from an *obsolete* 5.25 inch floppy disk drive's track selector motor. This project's 200 step motor has 4 phases, for a total of 5 wires when including the common or ground wire. Each phase has a resistance of approximately 70 Ohms to common or 140 Ohms between any two phases. Thus, at 12 Volts, each phase will draw slightly over 170 mA.

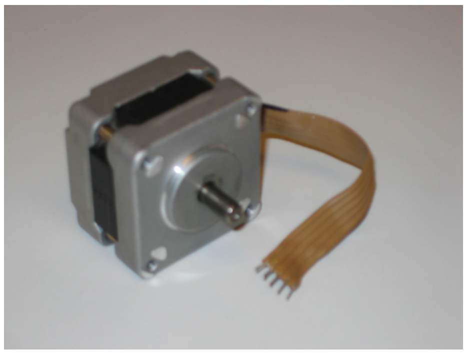

Figure 2: Stepping Motor

3.2 Stepping Motor Driver

A stepping motor driver is required to connect the low power microcontroller to the high power, low impedence motor. A simple driver was constructed from TIP31 NPN transistors in a TO-220 case. The transistors are connected with the common to the motor, base to the microcontroller, and the emmitter to ground. Each transistor is capable of more than about 500mA continuous current. The transistors' data sheet is given in the appendix. Referring to the schematic (Fig. 1), diodes placed across each driver circuit to prevent back EMF or motor generated current from burning out the transistors. Total driver cost is approximately $2.

An LED array is used to indicate the active motor phases. Four LEDs are connected to each of the driver outputs. 470 Ohm resistors restrict the current from a 5 volt input to a maximum of 10 mA. The LED's voltage drop improves the margin by even more.

3.3 Directional Gyro

The project's goal suggested using an actual DG, which was salvaged from an A&P mechanic's scrap pile. The vacuum powered DG, Part Number 23-600, was produced by Garwin in the 1960's[1] for Cessna[2].

The first step involved removing the existing mechanical gyro assembly and support structure. Because the selected stepping motor has only 400 discrete steps (0.45 degrees) per revolution using half-stepping, a 32:105 reduction drive connects the stepping motor output shaft to the DG's dial. The drive was designed and built by the author at no cost. The stepping motor's electrical input is through a 15 pin connector. The final electrically driven directional gyro is shown in Figure 3.

[1]The FAA stamp shows December 6, 1961.

[2]The Cessna logo is prominent on the display window

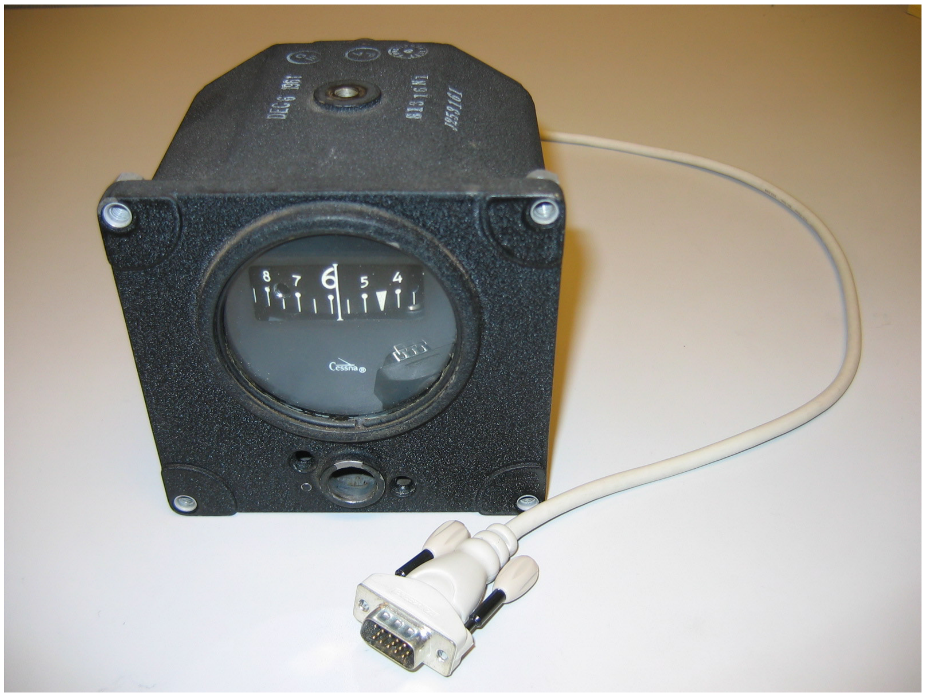

Figure 3: Converted Directional Gyro

3.4 Rate Gyro

Angular rates are sensed by an Analog Devices ADXRS150EB, which is the evaluation board for the 150 degree per second ADXRS150 gyroscope. From the datasheet[3], the rate gyro senses the Coriolis force resulting from angular rate and motion. A partial data sheet is in the appendix. The rate gyro chip requires 5 volts DC and outputs 12.5 mV per degree per second with a range of ± 150 degrees per second. A temperature compensation output was available but not used. The chip costs $50.

Figure 4 shows the analog voltage output for a known input motion. The gyro was tested with approximated, human generated input motions: ± 45 degrees per second, ± 90 degrees per second, two impulses, a chirp, and two small impulses.

3.5 Microcontroller

A PIC 16F876 microcontroller provided the signal processing and motor control. The micro-controller is a 28 pin DIP Microchip PIC16F876 clocked at 20 MHz. The PIC voltage input is +5 volts DC via a μA7805 dc/dc voltage regulator. The

[3]Available at www.analog.com

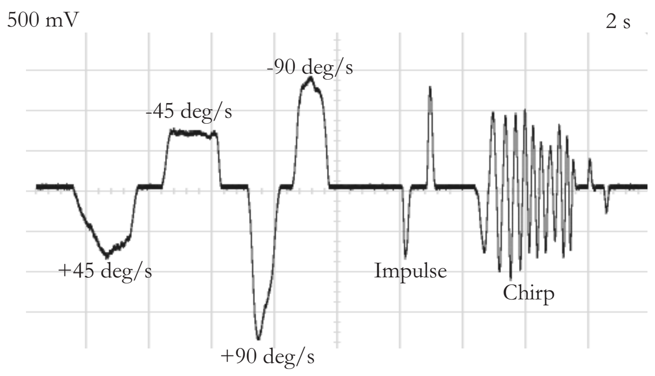

Figure 4: Rate Gyro Test

compiler is the CCS C compiler (v. 3.207) for 14 bit PIC chips. C compilation occurs on a x86 based PC. Data transfer between the PIC and the PC is through a 9 pin serial cable.

3.5.1 External Interrupt Button

The external button is connected to pin B0, the external interrupt pin. Pushing the single pole momentary button shorts pin B0 to ground. Otherwise, a hold-up resistor maintains about 5 volts on pin B0.

3.5.2 Calibration Circuit

A calibration circuit is connected to the PIC's A1 port. For improved calibration, a voltage divider circuit is used. The calibration will be within several millivolts of 2.5 volts, so two 5000 Ohm resistors[4] with a centered 360 Ohm potentiometer create a sensitive voltage adjustment.

4 Software

This section discusses the microcontroller software. The C source code is given in dg.c (p. 5) in the Code Listing section. The main() function contains a while loop with sequential motor updates and sensor readings. The external interrupt is used for an emergency shutoff switch.

4.1 Motor Control

The motor control software is contained in the `update_dial()` function. The function converts desired motor steps, both positive and negative, to motor

[4]Actually, one 5000 Ohm resistor is a 5000 Ohm potentiometer to account for resistor variance.

control outputs on the C port. The current motor position is necessary and is kept in the motor_pole_positions variable. Stepping motor phase and position correlation is specified in the positions array:

```
int positions[8]={0x01, 0x03, 0x02, 0x06, 0x04, 0x0C, 0x08, 0x09};
```

Half stepping is implemented; the native step locations are at hex 01, 02, 04, and 08; the half steps are at hex 03, 06, 0C. The `update_dial()` function moves the motor one step depending on the sign of the motor_pole_positions variable. The function returns the new motor pole positions.

4.2 Rate Sensing

Rate sensing and the strap down equation are acquired and calculated in the `velocity()` function. The rate gyro voltage is determined with a read_adc() function call.

Two important code lines are in this function. The first is the conversion of the rate sensor voltage to a heading change.

```
delta_heading += rate_sensor / SENSOR_VOLT_PER_DEG_S * (float) time *TIMER1_SCALE;
```

The second important code line converts the heading change to a motor position without dropping the non-integer part of the heading change. The residual headings are accumulated and added back into the delta_heading above; hence the `+=` statement.

```
residual_heading= modf(delta_heading*GEAR_RATIO, &delta_heading);
```

5 Improvements and Suggestions

The electronic directional gyro was implemented as described above. The final cost excluding the PIC chip is approximately $60. The DG worked and properly moved the DG dial to the correct heading. Equivalently, a pointer on the stepping motor remained in a constant inertial heading. However, more DG drift than expected was encountered because of the PIC's A/D conversion.

The following items were observed:

1. This project assumes no pitch nor roll motion. Thus this project is not a true Directional Gyro as would be useful for aircraft navigation.

2. The PIC analog to digital conversion is noisy with a variance *much* larger than the rate gyro's. Additionally, the PIC's noise appears non-Gaussian, impulsive, and periodic. Attempts to filter the rate gyro failed, because the noise is coming from inside the PIC chip.

3. The PIC 16F876 was not designed for a critical sensor application and was a poor choice for the microcontroller. Choosing an off-board A/D converter would be preferable. Also, the current PIC chip does not have enough floating point power to implement efficient digital filtering.

4. Time accuracy is not trivial in real-time sensor applications. Proper conversion to and from a floating point heading was essential.

5. Regardless of the rate gyro's sophistication, the system will drift. From theory, integration from rate to angle will resemble a random walk process when assuming no bias. The angular error's variance will approach infinity as time approaches infinity.

6. Using a stepping motor for an angle application by default gives discrete angles. A regular dc motor with an encoder would give *mostly* continuous angles with the extra advantage of allowing position feedback.

7. Adding an electronic magnetic compass would allow for a Kalman filter to be added. Magnetic compass errors during turns, accelerated flight, etc. would be mitigated with a small gain (assuming large magnetic variances).

References

LM324 Operational Amplifier DataSheet http://onsemi.com
ADXRS150 Yaw Rate Gyroscope http://www.analog.com

Code Listing

dg.c

```
/*
*       dg.c --- Directional Gyro
*
*       Charles O'Neill
*       MAE 5483
*       Term Project 2
*/

/*--------------------------------------------
* Default PIC Initilization
*--------------------------------------------*/
#include <16F876.h>
#device ADC=10
#include <math.h>
#use delay(clock=20000000)
#fuses HS,NOWDT
#use rs232(baud=19200,parity=N,xmit=PIN_C6,rcv=PIN_C7)

/*--------------------------------------------
* Function Prototypes
*--------------------------------------------*/
signed int16 update_dial(signed int16 number_of_steps);
signed int16 velocity(void);

/*--------------------------------------------
* Global Definitions
*--------------------------------------------*/
#define OFF                         0x00
#define MOTOR_STEP_DELAY            1
#define ADC_RES                     5.0/1024.0
#define SENSOR_VOLT_PER_DEG_S       0.0125
#define TIMER1_SCALE                 1.6E-6
#define GEAR_RATIO                  (400.0/360.0)

/*--------------------------------------------
* Global Variables
*--------------------------------------------*/
int1 state=1;
int1 sample_trigger;
signed int16 motor_steps_desired=0;
int16 motor_pole_position=0;
float heading, heading_motor, delta_heading;
int16 calibrate;
float volt_calibrate=2.5;
```

```
/*------------------------------------------------------------
* External Interrupt
*----------------------------------------------------------*/
#int_ext
void button_int(void){
   state^=1;
   delay_ms(50);
}

/*------------------------------------------------------------
* Main Program for calculation timer
*----------------------------------------------------------*/
void main(){

   /* Setup Analog to Digital Conversion */
   setup_adc_ports( ALL_ANALOG );
   setup_adc( ADC_CLOCK_DIV_32 );
   set_adc_channel( 0 );

   /* Setup Timer */
   setup_timer_1( T1_INTERNAL | T1_DIV_BY_8 );

   /* Initialize Interrupts */
   enable_interrupts(INT_EXT);                        // Port B Interrupt
   enable_interrupts(GLOBAL);                         // Turn on Interrupts

   /* Initialize Heading */
   motor_steps_desired=0;
   heading=heading_motor=0;

   /* Velocity Tracking */
   state=1;
   while(state){
      motor_steps_desired=update_dial(motor_steps_desired);
      delay_ms(MOTOR_STEP_DELAY);
      motor_steps_desired += velocity();
   }

   /* Emergency Out */
   state=1;
   output_c(OFF);
}

/*------------------------------------------------------------
* Sample Rate Sensor
*----------------------------------------------------------*/
signed int16 velocity(void){
   signed int16 motion, time;
   int16 sensor;
   float rate_sensor, residual_heading;
```

```
    /* Read Sensor Rate */
    sensor = read_adc();
    time=get_timer1();
    set_timer1(2);

    /* Calibration */
    set_adc_channel( 1 );
    calibrate = read_adc();
    volt_calibrate= (float) calibrate * ADC_RES;
    set_adc_channel( 0 );

    /* Update heading */
    rate_sensor= (float) sensor * ADC_RES - volt_calibrate;
    delta_heading += rate_sensor / SENSOR_VOLT_PER_DEG_S * (float) time *
        TIMER1_SCALE;

    /* Convert to integer */
    residual_heading= modf(delta_heading*GEAR_RATIO, &delta_heading);
    motion = (signed int16) (delta_heading);

    delta_heading=residual_heading;
    return(motion);
}

/*--------------------------------------------
* Update Dial
*--------------------------------------------*/
signed int16 update_dial(signed int16 steps){

    /* Initialize */
    int positions[8]={0x01, 0x03, 0x02, 0x06, 0x04, 0x0C, 0x08, 0x09};

    motor_pole_position%=8;

    /* Rotate Clockwise */
    if(steps>0){
        output_c(positions[--motor_pole_position %8]);
        return(--steps);
        /* Rotate Counter Clockwise */
    } else if(steps<0){
        output_c(positions[++motor_pole_position %8]);
        return(++steps);

    /* No Rotation */
    } else {
        output_c(positions[motor_pole_position %8]);
        return(steps);
    }
}
```

Data Sheets