University of Florida
Department of Electrical and Computer Engineering
EEL 5666
Intelligence Machine Design Laboratory

Autonomous Vertical Flight Control System for Helicopters

Projects Name: Cobra
Designer: Willie Hazin
Instructor: Dr. A. A Arroyo
Date: December 9, 1998

Table of Contents

ABSTRACT

The paper illustrates a control scheme for application to helicopters in vertical flight mode -- take-off, landing, and hovering -- to guarantee altitude stabilization. A nonlinear helicopter model is used to derive the proposed control system in which a Motorola MC68HC11 micro-controller will be used to establish the overall system stability. A 2-axis tilt sensor will be used to measure the angular rate and tilt of the main shaft for the feedback circuitry to the micro-controller. The final result of the proposed technique will allow the pilot to have two different controlling methods. The internal circuitry of the design will allow the pilot to switch from micro-controlled to radio controlled piloting through the use of an extra channel on the handheld transmitter. This will serve as a safety mechanism for the pilot in flight and for testing procedures.

EXECUTIVE SUMMARY

Due to the multidirectional movement capability of helicopters, the control scheme becomes much more difficult. Unlike standard airplanes, the helicopter movement depends on rotation and pitch of the main and tail rotor airfoil blades. The rotation of these blades causes the dynamics of the flight control scheme to become much more difficult. One of the maim problems that occurs due to the rotation of the main blades is that the fuselage of the helicopter tends to rotate in the direction rotation of the main blades. The method that will be implemented in this design scheme will minimize the problem by controlling vertical flight only, takeoff, hovering, and landing. The mechanics of the model helicopter that will be used to implement this control scheme will be controlled by a Motorola MC68HC11 micro-controller. Although the flight control of helicopters is a three-dimensional problem, The control scheme will be simplified to a three one-dimensional problem running simultaneously. Due to the 2 MHz speed of the micro-controller, the control program executes slower than the helicopter mechanics need to counter act any unstabilized movements. Another problem that the control program did not take into consideration due to the speed of the micro-controller is the calculation of the tilt angular acceleration. This needs to be done in order to insure that the helicopter reaches its stabilized point with zero momentum. I recommend a micro-controller that is at least five times the speed of the MC68HC11 in order for this to be taking in consideration.

INTRODUCTION

In the past few years, autopilots have become a standard device in all new commercial planes and even private planes. On the other hand, autopilots for helicopters have yet to be mastered. Due to the wide use of helicopters for private and commercial use, autopilots in all helicopters should be a standard device. This will decrease the number of personnel required in the helicopter to due a certain assignment allowing the pilot to be free to do other tasks. Having the proposed objectives achieved-- take-off, landing, and hovering-- it will simplify the future design of a full autopilot system for helicopters. This will eventually lead to a global directional autopilot where the helicopter will maintain a certain path chosen by the pilot as currently done in airplanes.

INTEGRATED SYSTEM

The helicopter that will be used is a radio-controlled model helicopter. It is controlled through the use of an eight-channel Futaba transmitter and the MC68HC11. The model helicopter uses five of the channels to control the different mechanics that are on board. The varies channels are used as follow-

1. Tail rotor
2. Throttle/Collective
3. Roll

4. Pitch
5. Flight attitude

One extra channel will be used to switch between micro-controlled and radio-controlled piloting. This channel will be an on/off switch. The circuitry that will be used to do this task will use five multiplexers. Four will be connected in parallel to switch between the outputs from the 68HC11 and the transmitter/receiver signals. The on/off channel that allows the pilot to switch between the different control systems will control the fifth multiplexer. The tail rotor servo on the other hand, will always be controlled by the receiver since it is controlled by a separate gyroscope to stabilize the tail. Since the objectives are take-off, landing, and hovering, this will have no effect.

The final control spec is the amount of altitude. This will be controlled through a seventh channel that will be incrementally variable on the transmitter. The helicopter will be able to maintain the amount of height that is specified from this channel using *a fixed-rate* of altitude movement. In other words, the helicopter will move at this fixed-rate in take-off and landing. For example, if the helicopter was hovering at 10 feet above ground and the seventh channel was suddenly moved to full altitude, the helicopter will not clime to this altitude using its maximum climbing acceleration. In stead, it will use this fixed-rate to clime to that point in

height or descend if desired. This will allow the helicopter to stabilize itself at every increment in the throttle servo before ascending or descending to the appropriate height.

MOBILE PLATFORM

The platform will be a 60-size Miniature Aircraft Model Helicopter that is powered by a 23cc engine. The cockpit of the helicopter was rearranged to fit all the needed circuitry to achieve the desired objectives. The model has all the mechanical mechanisms needed to duplicate the movement and maneuvering of a real helicopter. This is a great advantage for duplicating the same design to a real helicopter. Furthermore, 20 to 30 feet cable was used to connect the engine to a handheld switch. This switch would connects the spark plug to ground when closed causing the engine to shout off in case the helicopter went into any undesired movement. This will also prevent any possible damages that could occur to the helicopter and people around the site during testing.

ACTUATION

The servos that were used to control the different mechanics on the helicopter are standard size Futaba servos Part number S9202. The transmitter Part number FP-8UHP that was used includes five of these servos. The servo specifications are as fallow:

Power requirement: 4.8V

Output torque: 69.5 oz-in
Operating speed: 0.22 sec/60 deg
Weight: 1.7 oz.

In the proposed design, the switching circuitry was proposed using relays. After extensive testing, it was conclude that the internal circuitry of the relays malfunctions due to the extensive amount of vibrations they encounter during flight. Therefore, the design has been modified using multiplexers. The following diagram describes the switching circuitry that was used.

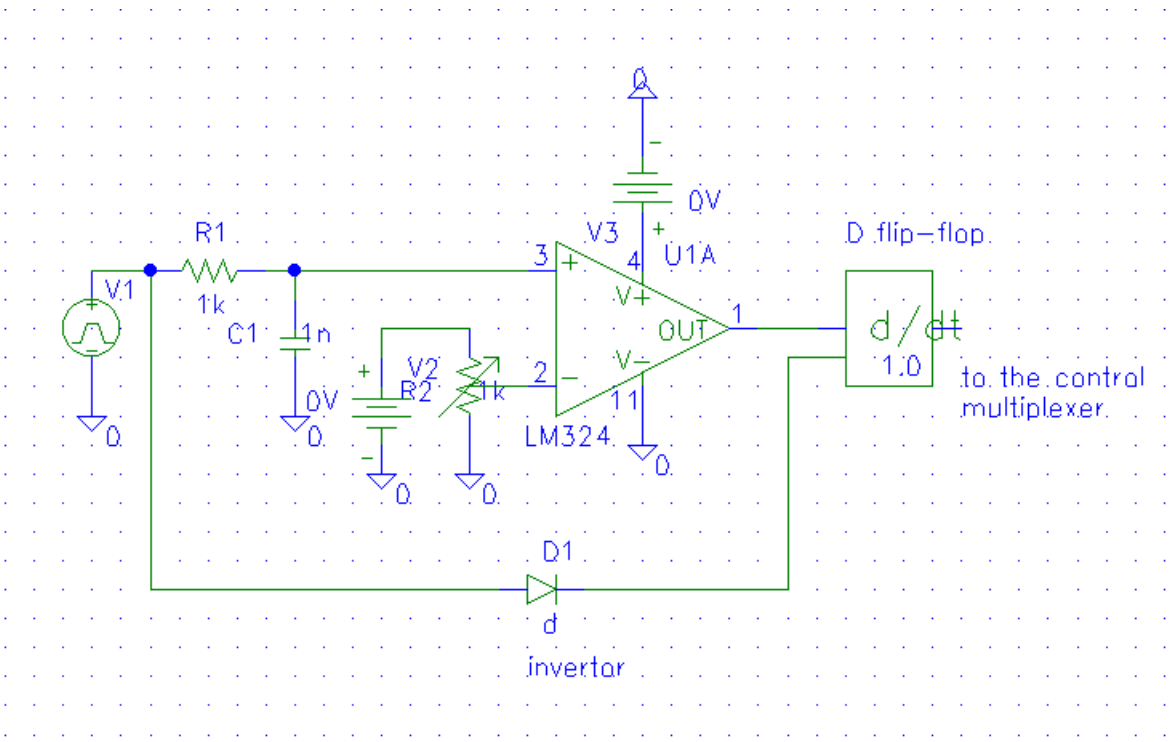


Figure 1-1, Switching Circuitry.

The input to the RC circuit is the pulse width from the receiver. This PWM signal has two different duty cycles depending on the position of the switch from the transmitter. This is what controls the input to the various servos. When the switch is on the duty cycle of the signal is bigger therefore causing the output of the integrator to have a larger peak voltage. This voltage is then compared to a fixed voltage set by a pot. If the voltage from the RC circuit is larger then the set voltage by the pot, the micro-controller inputs are then send to the various servos. If the voltage of the RC circuit is less than the fixed voltage by the pot, then the receiver inputs are send to the servos. The output of the D flip-flop is connected to the control multiplexer. The D flip-flop latches the input from the comparettor only on

the falling edge of the PWM signal. This ensures that the signal going to the control multiplexer only when the peak voltage of the RC circuit has been reached. The reset of the circuit is as follows:

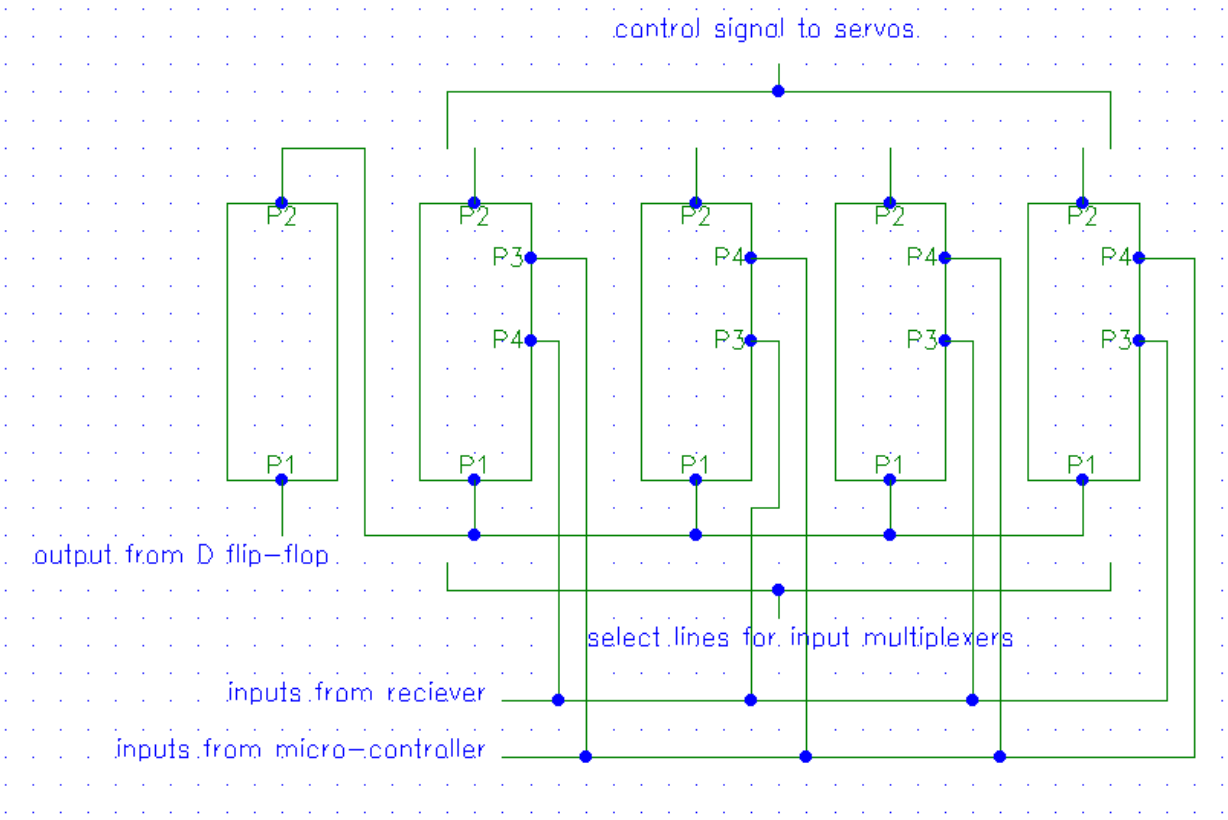


Figure 1-2, Muliplexer circuit for Switching Circuitry

SENSORS

Futaba Rate Gyro -- Model FP-G154--.This gyro will be used to stabilize the tail movement of the helicopter. There will be no output reading manipulation done from this gyro since it works in conjunction with the receiver. In other words, this gyro will be on in both control systems, manual and micro-controlled.

Push-button Switch. The switch will be connected to one end of a 20-feet cable while the other end of the cable is connected to the spark plug of the engine and ground. This switch will serve as a safety mechanism in case of undesirable movements of the helicopter while in flight causing the engine to turn off if the switch is closed.

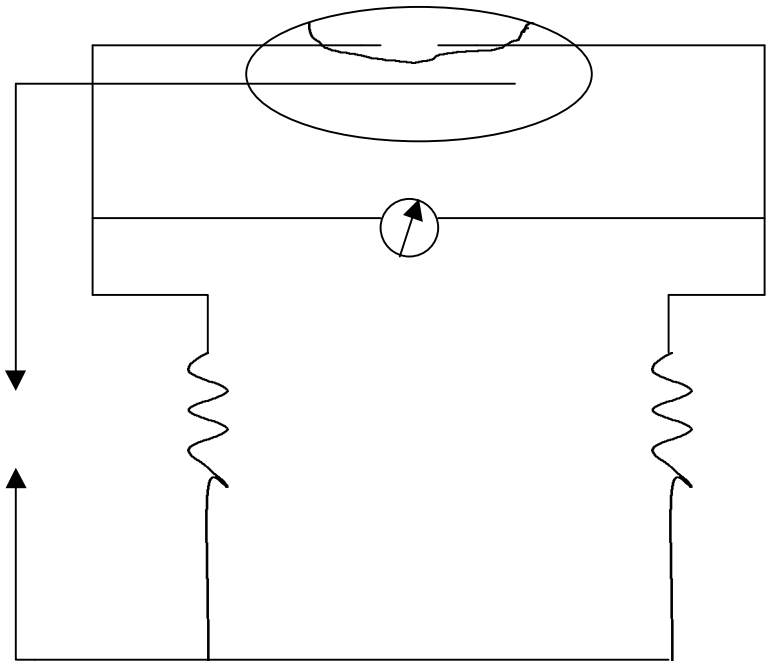
Tilt Sensor: The sensor used is a *Frederick's Company* 2-axis tilt sensor Part number 0729-1703-99 and it will be mounted on Part number 0717-2203 mounting board. The 2-axis tilt sensor will be used to ensure the pitch and roll in the plane in which the helicopter will be stabilizing itself. This sensor was also be used to measure the angular rate for the feedback circuitry to the micro-controller.

NOTE: This sensor will not be able to detect any movement in drift.

PRINCIPLE OF OPERATION

The tilt sensor is an envelope glass partially filled with electrolytic fluid, which contacts metal electrodes. It provides an output voltage proportional to the tilt angle when connected in an appropriate electrical circuit. Desired operating characteristics can be achieved by varying envelopes, electrolytes, and electrode configurations. When the sensors are connected as shown in the test circuit (Figure1-3), and leveled, an equal impedance to the common electrode will exist and the digital voltmeter will indicate a minimum output. Tilting the level will cause an unbalanced impedance to the common electrode and the output voltage

will increase. This voltage is the usable output of the sensor and is proportional to the tilt angle.



NOTE: An AC to DC conversion circuit is used in conjunction with the above circuit for a DC output.

SPECIFICATION OF THE 0717-2203 SENSOR BOARD

Input Voltage	7 Volts DC
Output Voltage	0 to 5 Volts
Operating Range	+ or - 30 degrees
Output Sensitivity	83.3 mV/degree
Null Voltage	2.5 Volts
Impedance At Null	400 ohms
Operating Temperature	25 C, + or - 15 C

Table 1-1, Board Specifications.

APPLICATION

The sensor board is mounted near the main axis of the helicopter where it will allow the sensor to produce maximum accuracy. The sensor board was configured in such a way that the zero degree output voltage is 2.5 Volts. This was done to accommodate the A/D converter on the 68HC11 micro-controller. This way no amplification is necessary. At this point, a test program (Program 1-1) is written to test the effects of the sensitivity of the tilt sensor due to vibration. The program takes a reading from the sensor every quarter of a second for 20 minutes for each axis. This will help explain the amount of noise the sensor sees and the amount of tilt the helicopter goes through during hovering.

This program was then compiled and down loaded to the 68HC11. The helicopter was then flown in a hovering position for about 20 minutes. The data collected was then down loaded to a text file using an input command ("s") from the keyboard. This was done using *Hyper Terminal Capture Text feature. Microsoft Excel* was used for calculations and graphing of the data. The data values were not converted to degrees to show more accuracy. Since 2.5 volts is the zero degree mark, the equivalent decimal value is 127. Therefore anything above or below this mark will show a change in ether of the axis. The following equations will explain the

accuracy of the sensor and equivalent degree value for an incremental change in the decimal values.

Output sensitivity of the sensor

2.5 Volts / 30 Degrees = 83.3 mV

The A/D sensitivity of the 68HC11

5 Volts / 255 = 19.6 mV

This means that the A/D can not detect and change in voltage less than 19.6 mV.

The accuracy that can be achieved from the sensor is

19.6 mV / 83.3 mV/deg. = .235 degrees

This means that the A/D can detect a 0.235-degree change from the sensor and that every incremental change in the decimal value is 0.235 degrees. This accuracy is sufficient enough for the helicopter application.

For example, let's say that we have a reading of 135 from the A/D, then the equivalent degree value is:

(135 - 127) * 0.235 = 1.88 degrees

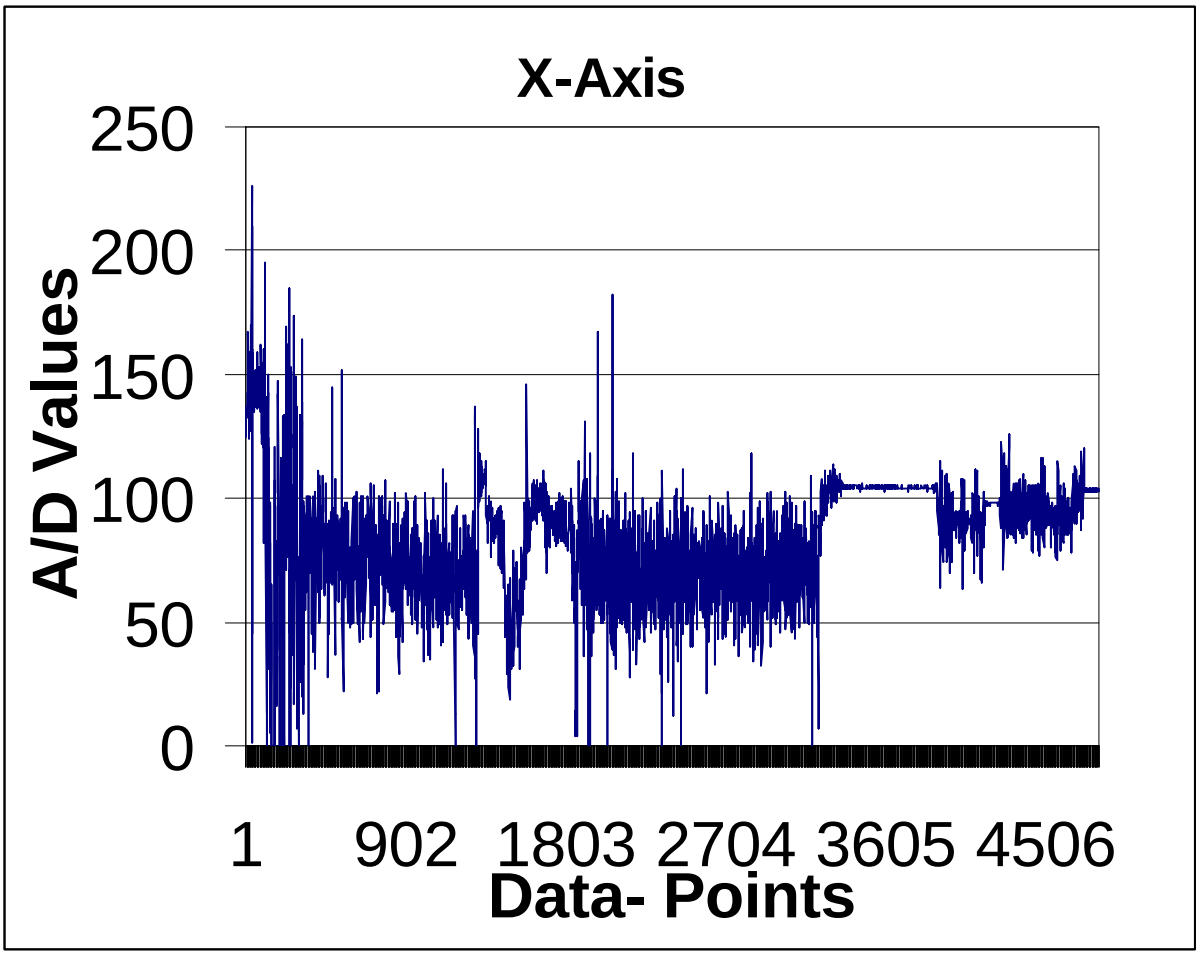


Table 1-2, The Roll Data Points.

The above graph shows 4800 data points collected while the helicopter was hovering. Although the graph looks like a lot of noise, it is extremely helpful in determining exactly where the helicopter sits in the "roll" position while hovering. Also its important to realize that the helicopter is moving in space while the "man" controller is trying to stabilize it in hovering. It's clear to see from the above graph that there are two possible set of data points that the helicopter sits in while hovering in the roll position and remembering that the data collected was over a 20-minute period. For the first 3600 points, its clear to see that the average is about

85 which is equivalent to about 10 degrees in roll. The second possible average point, which starts from 3600 and on, is about 95 and is equivalent to about 7.5 degrees. It is also important to realize that the sensor does not sit on the helicopter perfectly aliened with the center of gravity when it is suspended from its main shaft.

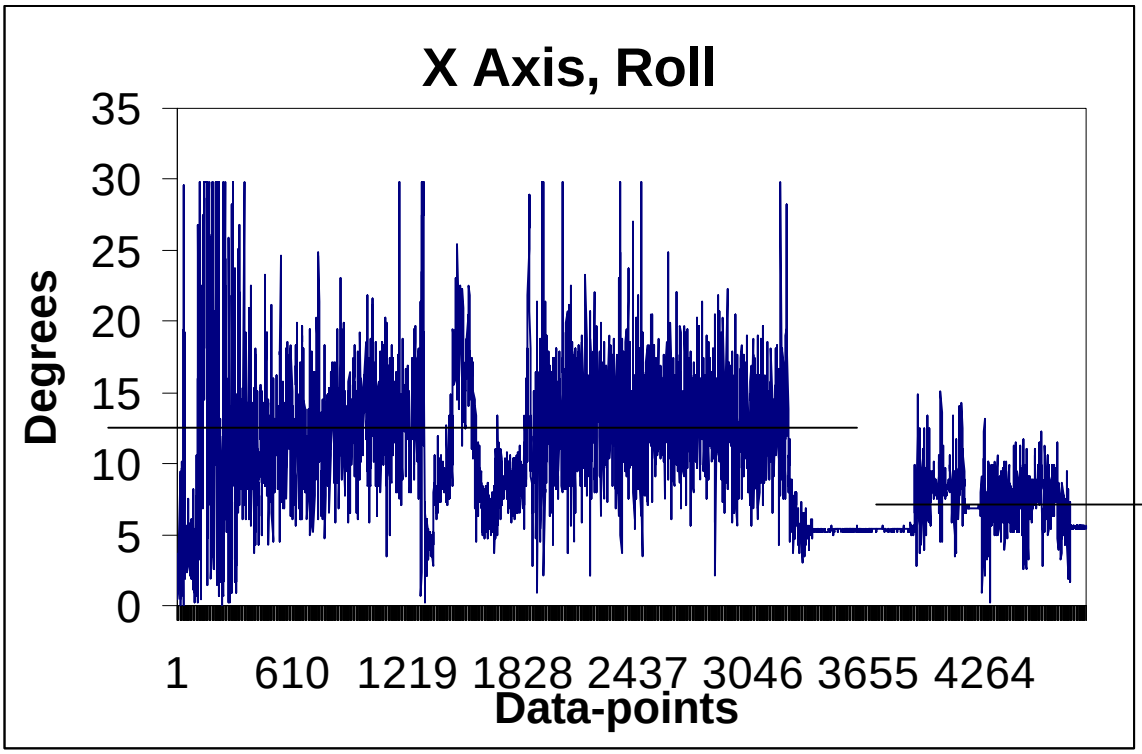


Table 1-3, Data points in Degrees for Roll.

The above graph shows the same data points as the previous graph except for the A/D values are converted to degrees using equations in the Application section. It's clear to see that the previous estimation was close within a few degrees. For the first set of data points, the average seems to be around 12.5 degrees instead of 10

degrees while the second set of data points seem to be around 7.5 degrees which is equivalent to the previous estimation.

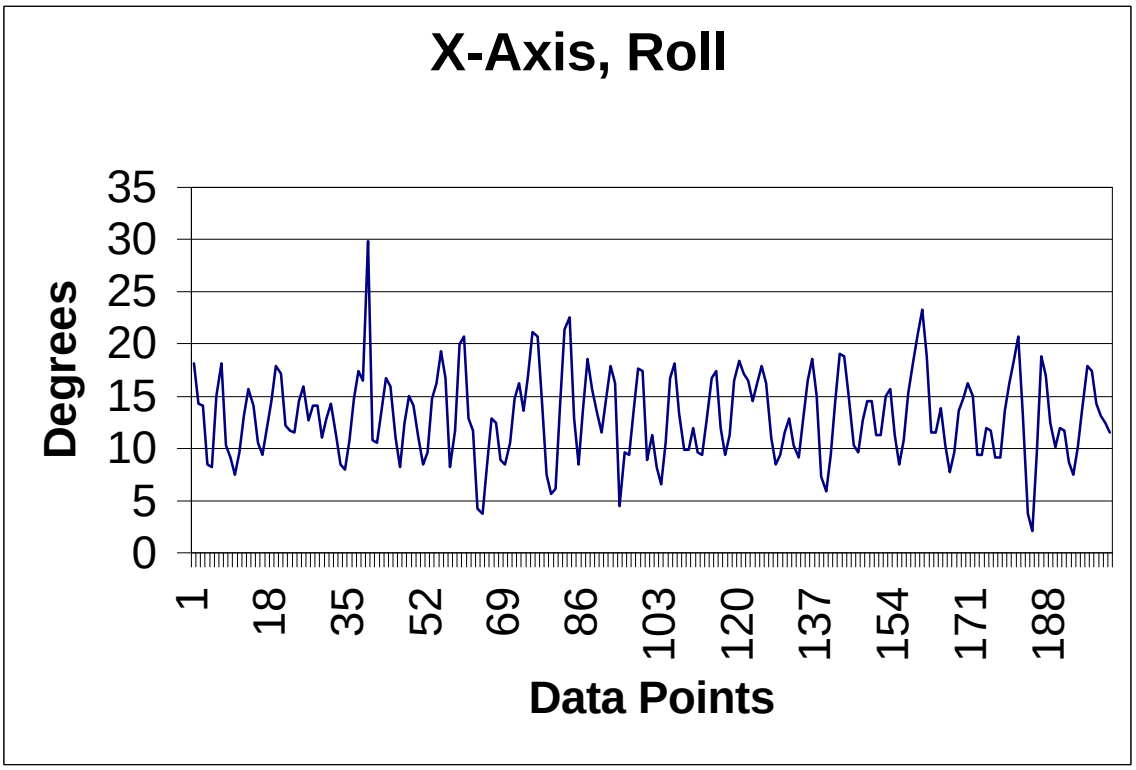


Table 1-4, Data points in Degrees for Roll.

The above graph is part of the previous graph except it's for 200 data points in the range of 2000 to 2200. Its equivalent time is approximately one minute. This range was efficient to select since it is in the middle of an extreme change while hovering. It shows the period of change in roll and the amount of degree change. The most important part of the above graph is that it shows the average amount of change from one point to another, which is approximately five degrees. The period of this change is about 3/4 of a second. Therefore, this graph explains the amount

of change in degrees per 3/4 of a second and tells the amount of fluctuation the helicopter goes through during hovering in the roll position.

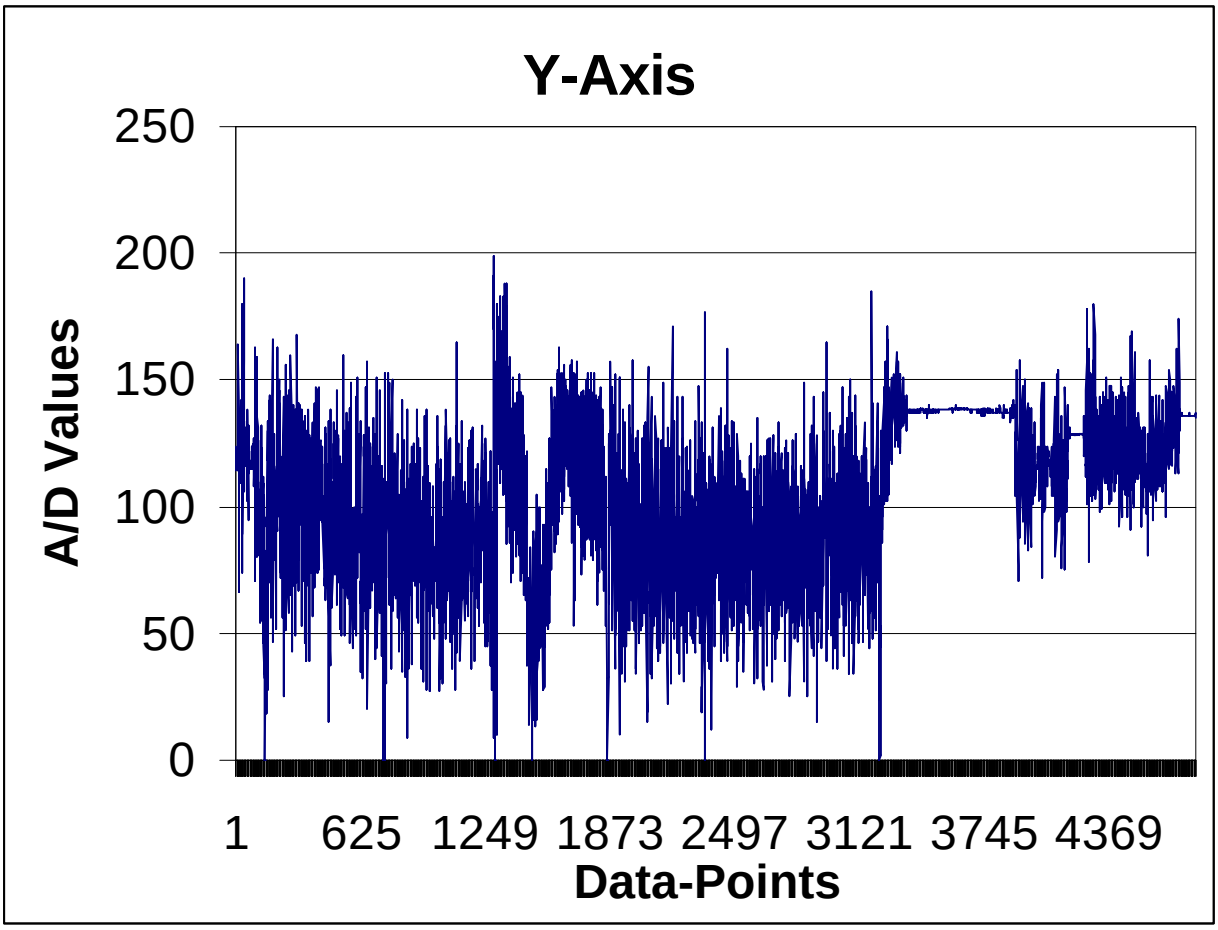


Table 1-5, Data point for the Pitch Axis.

This graph also shows 4800 data points collected while the helicopter was hovering. Although it also looks like a lot of noise, it is extremely helpful in determining exactly where the helicopter position sits in the pitch axis while hovering. Also this data is collected while the helicopter is moving in space while the "man" controller is trying to stabilize it in hovering. As it would be expected

from the graphs for the roll-axis in Table 1-2 and Table 1-3, there should also be two possible sets of points that the helicopter sits in while hovering in the pitch position. For the first 3200 points, its clear to see that the average is about 100 which is equivalent to about 7 degrees in pitch. The second possible average point, which starts from 3200 and on, is about 125 and is equivalent to about half a degree. It is also important to recall that the sensor does not sit on the helicopter perfectly aliened with the center of gravity when it is suspended from its main shaft.

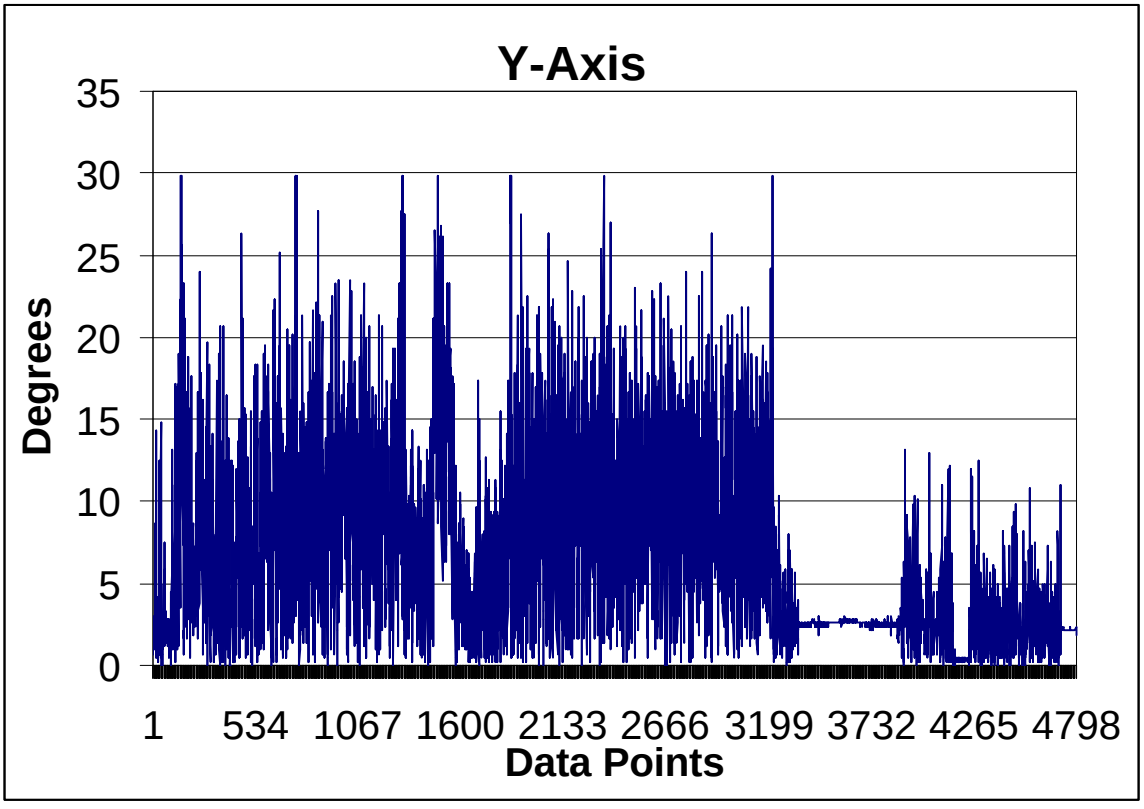


Table 1-6, Data Points in Degrees for Pitch.

The above graph shows the same data points as the previous graph except for the A/D values are converted to degrees using equations in the Application section. It's clear to see that the previous estimation was close within a few degrees. For the first set of data points, the average seems to be around 8 or 9 degrees instead of 7 degrees while the second set of data points seem to be around 2.5 degrees which is 2 degrees more than the previous estimation. This also goes to show that the calculations are accurate.

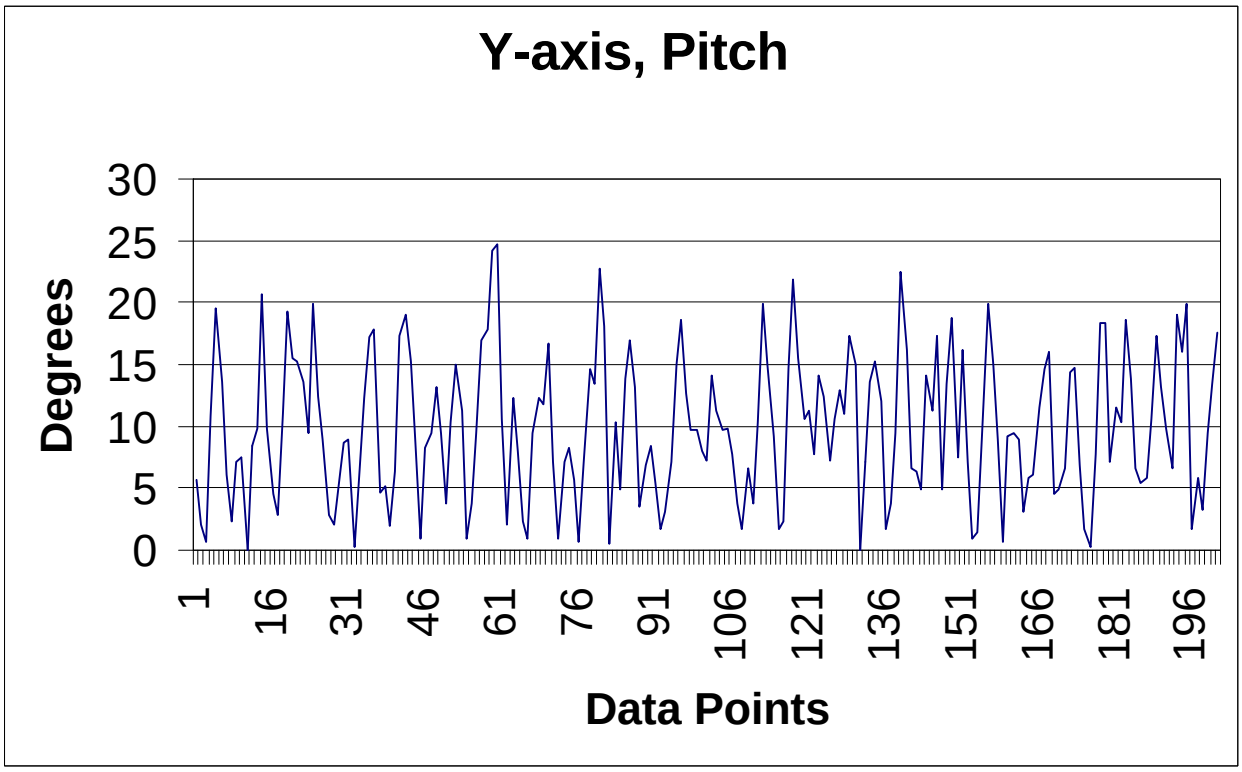


Table 1-7, Data points in Degrees for Pitch.

This graph is also part of the previous graph except it's for 200 data points in the pitch axis in the range of 2000 to 2200. Its equivalent time is approximately one

minute. This range is also efficient to select since it is in the middle of an extreme change for a hovering flight. The period change is about 6/10 of a second, which is close to the period change in the roll axis. But on the other hand, It's clear to see that the change in degrees is much higher than the change of degrees for the roll axis. This would be expected since there is more movement in pitch while the helicopter is trying to hover. The amount of degree movement for the 6/10 of a second period is about 15 degrees, which is two times higher than the roll movement. Therefore, this graph explains the amount of change in degrees per 6/10 of a second and tells the amount of fluctuation the helicopter goes through during hovering in the pitch position.

The above data has analyzed the sensor in specifications, sensitivity, and application to the helicopter movement. As for the two possible sets of data points in the graphs, it is evident to see that the first possible set of data points are the ones the helicopter experienced during hovering while the second set of points were collected while the helicopter is idling high on the ground since there was little change in degree movement.

It shows that vibration does have an effect on the sensitivity of the reading from the amount of noise the sensor experienced. From the data collected and the graphs plotted, its evident that some biasing has to be taking in affect for the control program. The data shows about 5 degrees of change for roll and about 10 degrees of change in pitch for a period of approximately one-second for each axis. This

peace of data is extremely helpful because it shows the amount of change the helicopter experiences during hovering and this will also be taking in affect in the control program. The next important peace of data, which is probably the most important peace, is the rate of change. This rate will be used in the control program to counter act the servo movements while the helicopter is stabilizing itself in flight.

BEHAVIORS

The behavioral algorithm that was used is based on experimental data collected during testing. The control algorithm program (Program 1-2) was written in ICC where the rate of change obtained from the sensor experiment was used to compensate any change in roll or pitch. The program is made up of multiple functions. The main program calls the height function first. This function checks the pulse width of the channel that was used for throttle control. The throttle duty (duty_t) is then compared to this duty cycle. Duty_t is initially set to the minimum possible duty that will keep the engine running. If throttle duty (duty_t) is less than the duty that was obtained from the height function, the program will increment or decrement duty_t until they are equal. The program also does not evaluate the inputs from the tilt sensor until the throttle duty is high enough for the helicopter to take off. This allows the helicopter not to adjust the pitch or roll servos until its time to take off the ground. Otherwise, the pitch or roll servos may be too far off the needed position when the helicopter starts to takeoff. Also the time required for

the servos to readjust themselves to the stabilization point may be too great for the specified rate the servos move on. The specified rate of the servo movement is 20 E clocks. This movement increases or decreases the duty cycle of the servos pulse width by about 1/100 of a millisecond. Although this may look like too small of an increment, it does make a difference in the servo movement since the duty cycle for servos vary from 2 to 4 ms. The sensor function is called after the height function although its values has no effect until the helicopters throttle reaches the takeoff point. The sensor function reads the A/D port, which is connected to the 2-axis tilt sensor, and stores 10 values of each axis in an array and sends the average of those values. These values are then compared to the stabilization values that were obtained from suspending the helicopter from its main shaft and reading the analog output of the sensor. This was critical to do since the sensor was not perfectly aliened with the center of gravity mounted on the helicopter. The program keeps running while changing the duty cycle of the various servos until it reaches its stabilized point. The motor interrupt function run every 32 ms. This ensures whatever duty value was last computed in the main program, it will be send to the appropriate servo at a constant rate.

CONCLUSION

The servo movements due react to the sensing of the 2-axis tilt sensor. This goes to show that the hardware and software are communicating as the design specifications call for. The switching circuitry of the relays did not function appropriately when the helicopter engine was running. The only conclusion attained from this was that the internal circuitry of the relay was malfunctioning, specifically the internal switch may have been bouncing due to the vibration of the engine. When the circuit was redesigned using muliplexers, it functioned appropriately. The MC68HC11 speed capability was not enough to compute all the necessary functions and still move the servos within an appropriate time to counter act any unstabilized movement of the helicopter. The size of the model used should also be increased so that wind effect is minimized. This tends to be a problem whenever the helicopter is ran during a gust of wind.

I feel that this project will give a better understanding of the complexity of stabilizing any nonlinear object in flight. The control design for the helicopter model presented here is based on a two DOF (degrees of freedom) scheme. The next step would be to apply this design technique to the four DOF helicopter model and eventually to the more complicated six DOF model.

Appendix

The following programs are written in ICC11.

```
/*****************************************************************************
*****************************************************************************
* programmer: Willie Hazin
* Version 1
*
*/
/*
#include <servotj.h>
#include <irtj.h>
*/
#include <analog.h>
#include <vectors.h>
#include <serial.h>
/*****************************************************************************
****/

int main(void)
{
    int count1, count2;
    int datax[4800];
    int datay[4800];
    int ch = 0;
    init_analog();
    init_serial();
    write("working");

        for(count1 = 0; count1 < 4800; count1++)
        {
                datax[count1]=analog(1);
                datay[count1]=analog(2);
            for(count2 =0; count2 < 2500; count2++);
        }
```

```
        while(1)
        {
        ch = get_char(); if (ch=='s')
        break;
        }
        for(count2 = 0; count2 < 4800;count2++)
        {
                write_int(datax[count2]);
        }
        write("y-axis");
        for(count2 = 0; count2 < 4800 ;count2++)
        {
                write_int(datay[count2]);
        }
}
```

Program 1-1,Sensor Test Program.

```
/* Title        Control.c
 * programmer       Willie hazin
 * Date         11/25/98
 * Version    1
 * Description
 * this program will control all the required procedures needed to control the a helicopter in
 * flight. It will control the pitch, Roll, height, and the power on board. power include gas
 * and battery supply. The height will be controlled based on an input from the transmitter.
 */
/***************** Includes ************************************/
#include <hc11.h>
#include <mil.h>
#include <subvect.c>
#include <me11.h>
#include <serial.h>
#include <analog.h>

/***************** Constants ***********************************/
#define PERIODM                 30500
#define MAX_CNT                 0xFFFF

#pragma interrupt_handler motor0 motor1 motor2 motor3
void motor0();
void motor1();
void motor2();
void motor3();

#define       BIT7  0x80  /* 10000000 */
#define BIT6        0x40  /* 01000000 */
#define BIT5        0x20  /* 00100000 */
#define       BIT4  0x10  /* 00010000 */
#define BIT2        0x04  /* 00000100 */
#define BIT1        0x02  /* 00000010 */
#define BIT0        0x01  /* 00000001 */
#define       BIT54         0x30  /* 00110000 */
#define INV6        0xBF /* 10111111 */
#define INV5        0xDF /* 11011111 */
```

```
#define INV4        0xEF  /* 11101111 */
#define INV1        0xFD /* 11111101 */
#define INV0        0xFE  /* 11111110 */

/********************** Data Global Variables ************************/
int duty_cycle[4];  /* Specifies the PWM duty cycle for two motors */
int duty_t, duty_x, duty_y; /* duty's that will be sent to the servos */
unsigned int x_axis, y_axis;
unsigned int          duty;  /* duty from the input capture */

/*************** Functions ***************************************/
void height(void)
/* Function: This routine determines the duty ratio for the throttle and the main
blades servos
 * Inputs:   None
 * Outputs:  duty (global)
 * Notes:    None
 */
{
int over, i, i2, i3;
unsigned int duty_array[10];
unsigned int first1, last1, total;
        i=0;
        for (i3 = 0; i3 < 10;i3++)
        {
                over = 0;                       /* set the overflow counter to zero */
                TCTL2 = TCTL2 | BIT0;           /* capture on rising edge  */
                TCTL2 = TCTL2 & INV1;
                TFLG1 = BIT0;                   /* clear the capture flag */
                TFLG2 = BIT7;               /* clear the timer overflow flag */
        while (!(TFLG1 & BIT0));                /* expression is true when edge */
                                        /* has NOT been detected */
                first1 = TIC3;
/*              TFLG1 = BIT0;                    clear IC3 flag   */
                TCTL2 = TCTL2 | BIT1;           /* capture on falling edge  */
                TCTL2 = TCTL2 & INV0;
                TFLG1 = BIT0;                   /* clear the capture flag */
        while (!(TFLG1 & BIT0));                /* expression is true when edge */
                 if (TFLG2 & BIT7)              /* has NOT been detected */
                 {
                        over++;
```

```
                TFLG2 = BIT7;     /* clear the TOF */
              }
            last1 = TIC3;
            if (last1 > first1)             /* calculate the duty ratio */
                {
                duty_array[i] = last1 - first1;
                i++;
                }
            else
                {
                duty_array[i] = (MAX_CNT+1) - first1 + last1;
                i++;
                }
            if (i == 10)
            {
                total=0;
                for (i=0; i<10; i++)
                {
                    total = total + duty_array[i];
                }
                duty=total/10;
            }
            for (i2=0; i2<10000; i2++);
        }
}
/*************** Functions ****************************************/
void init_motors(void)
/* Function: This routine initializes the motors
 * Inputs:   None
 * Outputs:  None
 * Notes:    This routine MUST be called to enable motor operation!
 */

{
  write("in init_motors");
  INTR_OFF();

/* Set all OCx pins to output low */
  SET_BIT(TCTL1, 0xAA);
  CLEAR_BIT(TCTL1, 0x55);
```

```
/* Set PWM duty cycle to intial settings */
  duty_cycle[0] = duty_t = 2300;
  duty_cycle[1] = duty_t;
  duty_cycle[2] = duty_y = 3000;
  duty_cycle[3] = duty_x = 3000;

/* Enable OC5 from PACTL */
  CLEAR_BIT(PACTL, 0x04);

/* Enable motor interrupts on OC2,OC3,OC4,and OC5 */
  SET_BIT(TMSK1, 0x78);

  INTR_ON();
        write("out init_motors");
}
void motor(int index, int per_cent_duty_cycle)
/* Function: Sets duty cycle and direction of motor specified by index
 * Inputs:   index in [0,1,3 ,4]
 *           per_cent_duty_cycle
 *
 * Outputs:  duty_cycle[index]
 *
 * Notes:    Checks for proper input bounds
 */
{

  duty_cycle[index] = per_cent_duty_cycle;

}

void motor0()
/* Function: This interrupt routine controls the PWM to motor0 using OC2
 * Inputs:   duty_cycle[0] (global)
 * Outputs:  Side effects on TCTL1, TOC2, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
/* Keep the motor off if no duty cycle specified.*/

  if(duty_cycle[0] == 0)
```

```
  {
   CLEAR_BIT(TCTL1, 0x40);
  }
 else
  if(TCTL1 & 0x40)
   {
    TOC2 += duty_cycle[0];              /* Keep up for width */
    CLEAR_BIT(TCTL1, 0x40);             /*  Set to turn off */
   }
  else
   {
    TOC2 += (PERIODM - duty_cycle[0]);
    SET_BIT(TCTL1, 0x40);               /* Set to raise signal */
   }
 CLEAR_FLAG(TFLG1, 0x40);                    /* Clear OC2F interrupt Flag */
}

void motor1()
/* Function: This interrupt routine controls the PWM to motor1 using OC3
 * Inputs:   duty_cycle[1] (global)
 * Outputs:  Side effects on TCTL1, TOC2, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
 /* Keep the motor off if no duty cycle specified.*/

 if(duty_cycle[1] == 0)
  {
   CLEAR_BIT(TCTL1, 0x10);
  }
 else
  if(TCTL1 & 0x10)
   {
    TOC3 += duty_cycle[1];              /* Keep up for width */
    CLEAR_BIT(TCTL1, 0x10);             /*  Set to turn off */
   }
  else
   {
    TOC3 += (PERIODM - duty_cycle[1]);
    SET_BIT(TCTL1, 0x10);               /* Set to raise signal */
```

```
    }
  CLEAR_FLAG(TFLG1, 0x20);                    /* Clear OC3F interrupt Flag */
}

void motor2()
/* Function: This interrupt routine controls the PWM to motor1 using OC4
 * Inputs:   duty_cycle[1] (global)
 * Outputs:  Side effects on TCTL1, TOC4, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */
{
 /* Keep the motor off if no duty cycle specified.*/

  if(duty_cycle[2] == 0)
   {
    CLEAR_BIT(TCTL1, 0x04);
   }
  else
    if(TCTL1 & 0x04)
     {
      TOC4 += duty_cycle[2];              /* Keep up for width */
      CLEAR_BIT(TCTL1, 0x04);                /*  Set to turn off */
     }
    else
     {
      TOC4 += (PERIODM - duty_cycle[2]);
      SET_BIT(TCTL1, 0x04);                /* Set to raise signal */
     }
  CLEAR_FLAG(TFLG1, 0x10);                    /* Clear OC4F interrupt Flag */
}

void motor3()
/* Function: This interrupt routine controls the PWM to motor3 using OC5
 * Inputs:   duty_cycle[1] (global)
 * Outputs:  Side effects on TCTL1, TOC5, TFLG1.
 * Notes:    init_motors() assumed to have executed
 */

{
 /* Keep the motor off if no duty cycle specified.*/
```

```
  if(duty_cycle[3] == 0)
   {
    CLEAR_BIT(TCTL1, 0x01);
   }
  else
    if(TCTL1 & 0x01)
     {
      TOC5 += duty_cycle[3];              /* Keep up for width */
      CLEAR_BIT(TCTL1, 0x01);               /*  Set to turn off */
     }
    else
     {
      TOC5 += (PERIODM - duty_cycle[3]);
      SET_BIT(TCTL1, 0x01);               /* Set to raise signal */
     }
  CLEAR_FLAG(TFLG1, 0x08);                    /* Clear OC5F interrupt Flag */
}
void sensor()
/* Function: This interrupt routine reads the tilt sensor from the analog port
 * Inputs:   None
 * Outputs:  the digital values from the x-axis and y-axis
 * Notes:    None
 */
{
    int count1, count2, totalx, totaly;
    int datax[10];
    int datay[10];
    int ch = 0;

        for(count1 = 0; count1 < 10; count1++)
        {
                datax[count1]=analog(1);
                datay[count1]=analog(2);
            for(count2 =0; count2 < 2500; count2++);
        }
        totalx = totaly = 0;
        for(count2 = 0; count2 < 10; count2++)
        {
                totalx = (totalx + datax[count2]);
                totaly = (totaly + datay[count2]);
        }
```

```
        x_axis = totalx / 10;
        y_axis = totaly / 10;
}

void main()
{
int     count4;
init_analog();
init_serial();
init_motors();
        while (1)
        {
                sensor();
                height();
        if (duty_t > 2850)              /* to make sure the heli. is off the ground */
        {                               /* duty_t is for the throttle and blades */

                if (x_axis < (135-3))
                {
                        duty_x=duty_x - 20;
                        motor(3, duty_x);
                }
                if (x_axis > (135+3))
                {
                        duty_x=duty_x + 20;
                        motor(3, duty_x);
                }
                if (y_axis < (129-3))
                {
                        duty_y=duty_y + 20;
                        motor(2, duty_y);
                }
                if (y_axis > (129+3))
                {
                        duty_y=duty_y - 20;
                        motor(2, duty_y);
                }
        }
      for (count4=0; count4<10000; count4++);  /* delay before increasing to next
height */
                if (duty_t != duty )
```

```
            {
                if (duty_t < duty)   /* duty from the height function */
                {                    /* duty_t is for the throttle and blades */
                duty_t = duty_t + 20;
                motor(0, duty_t);
                motor(1, (6200-duty_t));
                }
                else
                {
                duty_t = duty_t - 20;
                motor(0, duty_t);
                motor(1, (6200-duty_t));
                }
            }
        }
}
```

Program 1-2, Control Program.