Department of Electrical and Computer Engineering
University of Victoria

ELEC 399
Design Project I
Final Report

Autonomous Quadcopter

Project Supervisor(s):	**Dr. Fayez Gebali**		
	Dr. Haytham Elmiligi		
Group Number:	**9**		
Website URL:	**http://web.uvic.ca/~jarrod/ELEC399/**		
Report Submission Date:	**December 3rd, 2012**		
Group:	**Jarrod Ferguson**	**V00708860**	**(jarrod@uvic.ca)**
	Taylor Coulthard	**V00708475**	**(tcoultha@uvic.ca)**
	Evgeny Schastlivenko	**V00714012**	**(geners@uvic.ca)**

Table of Contents

Table of Figures

1 Goals

Our ELEC399 Design Project was to design, build and program an autonomous quadcopter. That is, assemble and code a working 4-motor, 4-propeller copter that flies without use of a remote control.

1.1 Planning

In our initial meeting with Dr. Gebali and his associate Dr. Elmiligi, we were shown several design projects but the one that intrigued us the most was using a microcontroller to operate a quadcopter. In this meeting we were shown videos and discussed ideas for practical and advanced uses of this device which furthered our decision to pick this as our semester long project.

The next step was making a plan. The obvious starting point would be obtaining and assembling the various parts to the copter including electric motors, speed controllers and a microcontroller board. We were very pleased to find out that Dr. Gebali and Dr. Elmiligi had already purchased all the required parts and assembled the copter frame.

This allowed us to start right away on the second step of finding out how we would code the microcontroller and what language we would be coding in. After trying AVR – eXtreme Burner to load the hex files we decided to use a combination of AVR Studio and AVRdude to convert the C files to hex and load the hex files to the microcontroller.

Third we needed to outline what we wanted to accomplish in ELEC 399 and set to work starting test code for it. After carefully considering our options we settled on getting the copter to hover without external force for a defined period of time.

Lastly, we would need to write code for a final working copy of the copter that would demonstrate our progress over the term. Unfortunately we had to revise our final goal due to unforeseen issues, but we were able to get a working code that allows the copter to hover while being tied down.

1.2 Equipment Used

The parts to our quadcopter were ordered from China and the parts list includes:

- 4 extension arms with housing for an electric motor (~22 cm long)
- 2 connector plates to secure the extension arms (~10 cm x 10 cm)
- 4 brushless electric motors
- 4 brushless electric speed controllers (FMT 25 A)
  - ESC programmer
- 4 propellers with attaching bolts and screws (25 cm tip to tip)
- KKmulticontroler board with on board Atmega168PA microcontroller
- 3 cell Lithium-polymer battery 11.9 V output
- Battery charger
- Software:
  - AVR Studio 6
  - AVRDude

1.3 ELEC399 Goals & Deliverables

As stated before at the end of the ELEC399 semester we hoped to have the quadcopter able to hover in an unaided flight. To complete this we had a few deliverables that we would need to accomplish to achieve our larger goal of flight:

- Get one motor to turn by using pulse width modulation
- All 4 motors spinning in unison and lift the copter while tethered
- Be able to read the on board gyroscopes and adjust flight using that data
- Consolidating the 3 previous goals into a flying demo that can automatically stabilize itself

Turning 1 motor was always going to be the toughest task. We finally met this deliverable November 27th, 2012 after working through difficulties with pulse width modulation and the speed controllers.

Achieving lift with 4 motors is a very significant deliverable for us. With all the challenges we faced during this project it came very close to not flying at all. We met this deliverable November 30th, 2012.

Unfortunately we were unable to achieve 2 of these deliverables due to issues with hardware and software; these will be discussed later in this report.

1.4 ELEC499 Goals

In ELEC499 we would like to be able to expand upon what we learned this term by using the copter to perform the task of flying on a virtual tour of the University of Victoria with an attached camera to display to new students the locations of faculty specific buildings. Deliverables for this project would include the previous 2 points of using the gyroscopes to create a flying demo as well as:

- Using an external accelerometer to help define the position of the copter in the sky
- Attaching a camera to the copter and deciding on a method to deliver the footage
- Being able to perform short flying tasks to navigate known obstacles
- Being able to perform long flying tasks and returning to the same spot

2 Code Implementation

In this section the code that was used will be explained as well as explaining the outputs that are generated.

2.1 Loading .hex Files to the Microcontroller

As we wanted to write our code for the microcontroller in C, we needed a program to convert the C file to an appropriate hex file and then transfer the hex file to the microcontroller. The solution we settled on was writing the C code in AVR studio which will automatically make a corresponding hex file when the Rebuild Solution is clicked see Figure 2-1 for an example:

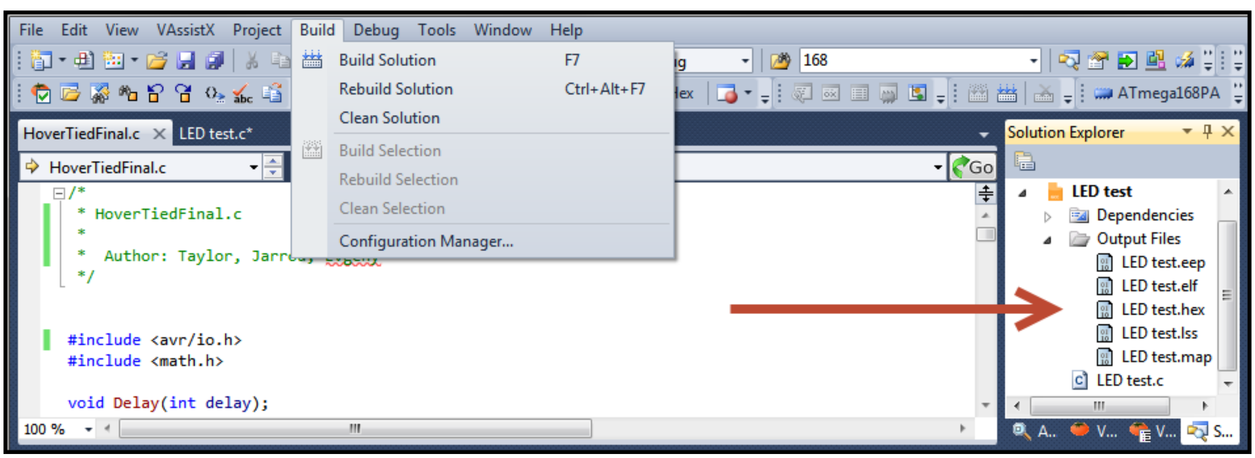

Figure 2-1: AVR Studio Example

We then needed to transfer the file to our microcontroller using a command line program called AVRDude. To allow the file to be transferred correctly in needs to be moved to the folder where AVRDude is and then navigate to the folder using command prompt. Here is an example of the commands required to run AVRDude from start to finish:

```
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Horn>cd desktop/

C:\Users\Horn\Desktop>cd avrdude-5.11-Patch7610-win32

C:\Users\Horn\Desktop\avrdude-5.11-Patch7610-win32>avrdude -c usbasp -p atmega16
8P -U flash:w:Tied1.hex:i

avrdude: warning: cannot set sck period. please check for usbasp firmware update
.
avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.07s

avrdude: Device signature = 0x1e940b
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed

         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: warning: cannot set sck period. please check for usbasp firmware update
.
avrdude: reading input file "Tied1.hex"
avrdude: writing flash (536 bytes):

Writing | ################################################## | 100% 0.37s



avrdude: 536 bytes of flash written
avrdude: verifying flash memory against Tied1.hex:
avrdude: load data flash data from input file Tied1.hex:
avrdude: input file Tied1.hex contains 536 bytes
avrdude: reading on-chip flash data:

Reading | ################################################## | 100% 0.32s



avrdude: verifying ...
avrdude: 536 bytes of flash verified

avrdude: safemode: Fuses OK

avrdude done.  Thank you.
```

Figure 2-2: AVRDude Example

2.2 1 Motor Turning

Turning just one motor was the most difficult part of this entire project. We incurred hardware and software setbacks as well as needing to learn how pulse width modulation works with our microcontroller.

```c
int main (void)
{
    DDRB  = 0b00000100;   // PB2 as an output

    TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM10);      // Non inverting mode on OC1A and OC1B, Mode Fast PWM 8-bit, Bottom Start, 0x00FF
    TCCR1B = _BV(WGM12) | _BV(CS11) | _BV(CS10);          // Clock/64

    OCR1B = (unsigned char)(0);
    Delay(4000);
    OCR1B = (unsigned char)(160);
    Delay(4000);
    OCR1B = (unsigned char)(0);
    Delay(4000);

    while (1)
    {
        OCR1B = (unsigned char)(160);
        }
    }

    return(0);
}
```

Figure 2-3: Spinning 1 Motor

To operate the motors we are unable to simply output a 1 or 0 to that port as it would either be full power or none. Instead we need to output a series of 1's and 0's in a square waveform at high frequency. This will output voltage dependant on how often the wave is on and off, called the wave's duty cycle in percentage. The method of doing this is called pulse width modulation (PWM) and there are 6 different PWM channels on our board. The first 3 lines of code in Figure 2-3 are used to initialize the PWM to output to port M1, we need to set port B pin 2 as an output as that is the location of M1 – see Figure 2-5. The next 2 lines are used to set up the timer/counter registers to enable pulse width output. Using the Atmega168PA datasheet we were able to find the values needed as in Figure 2-4 we needed to set the value of COM0A1 to 1 for our PWM. We found the values for all the other pins related to the timer/counter; they gave us our 2 register definitions for TCCR1A and TCCR1B.

COM0A1	COM0A0	Description
0	0	Normal port operation, OC0A disconnected.
0	1	WGM02 = 0: Normal Port Operation, OC0A Disconnected. WGM02 = 1: Toggle OC0A on Compare Match.
1	0	Clear OC0A on Compare Match, set OC0A at BOTTOM, (non-inverting mode).
1	1	Set OC0A on Compare Match, clear OC0A at BOTTOM, (inverting mode).

Figure 2-4: Values for COM0Ax

The rest of the code serves the purpose of initializing the speed controllers to "arm" them. A very common aspect of speed controllers is that they will not accept input immediately after they are turned on; they require a short period of time where the output is 0 before they will accept any speed values. Our speed controllers were a touch more tricky to arm: we needed to output a 0 value, followed by a value around the point it would turn on at (160 out of 255), and finally a 0 value again before they would arm correctly. After the arming phase is over we enter an infinite while loop with a set value for the PWM and the motor will spin.

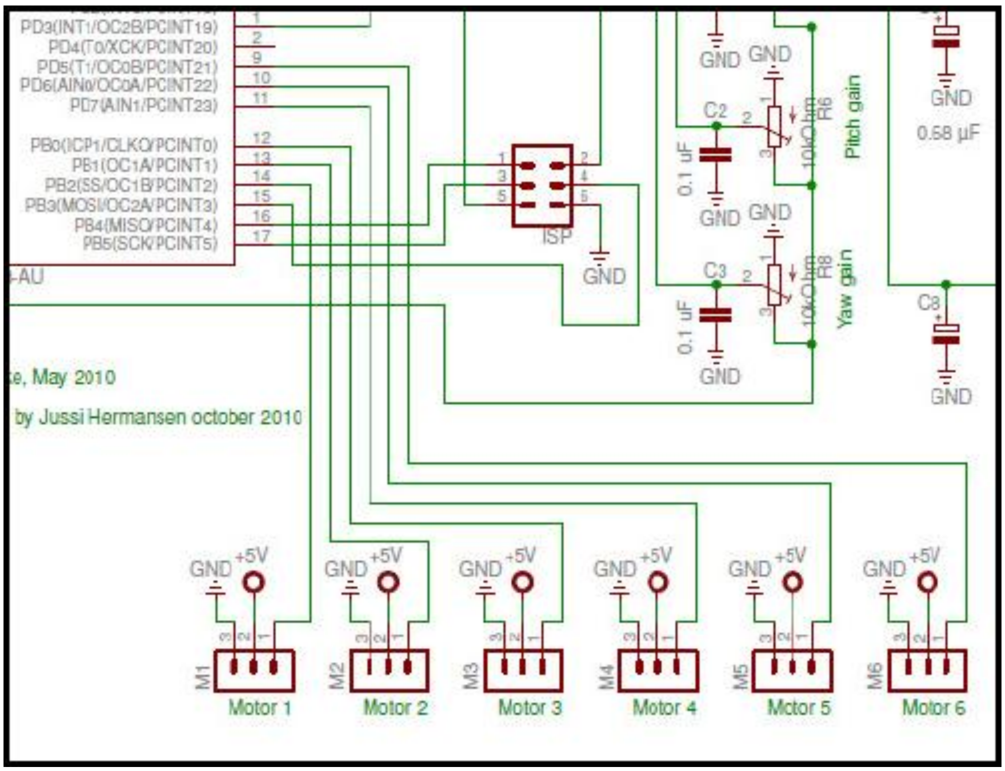

Figure 2-5: Pin Locations & Outputs

Another point of note is the Delay function:

```
void Delay(int delay)
{
    int x, y;
    for (x = delay; x != 0; x--)
    {
        for (y = 1000; y != 0; y--)
        {
            asm volatile ("nop"::); //do nothing for a bit
        }
    }
}
```

Figure 2-6: The Delay Function

After having some trouble with the delay function we were given we decided that we needed to try a different one. The function we settled on runs through 2 for loops for a given amount of “delays” when the function is called.

2.3 All 4 Motors with Delays and Acceleration/Deceleration

The code for our tests with propellers attached was essentially the same except that we had to take into account acceleration and deceleration to take off and land softly.

```
void Arm_ESC(void){
    DDRB  = 0b00000110;   // PB1 and PB2 as outputs
    DDRD  = 0b01100000;   // PD5 and PD6 as outputs

    TCCR0A = _BV(COM0A1) | _BV(COM0B1) | _BV(WGM00) | _BV(WGM01);      // Non inverting mode on OC2A and OC2B, Mode Fast PWM, Bottom Start, 0xFF
    TCCR0B = _BV(CS01) | _BV(CS00);       // Clock/64

    TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM10);      // Non inverting mode on OC1A and OC1B, Mode Fast PWM 8-bit, Bottom Start, 0x00FF
    TCCR1B = _BV(WGM12) | _BV(CS11) | _BV(CS10);          // Clock/64

    OCR1B = (unsigned char)(0);                          // Zero out all the outputs
    OCR1A = (unsigned char)(0);
    OCR0A = (unsigned char)(0);
    OCR0B = (unsigned char)(0);
    Delay(4000);
    OCR1B = (unsigned char)(160);                        // Arm motors by inputing lowest value
    OCR1A = (unsigned char)(160);
    OCR0A = (unsigned char)(150);
    OCR0B = (unsigned char)(150);
    Delay(4000);
    OCR1B = (unsigned char)(0);                          // Init to zero before accelerating
    OCR1A = (unsigned char)(0);
    OCR0A = (unsigned char)(0);
    OCR0B = (unsigned char)(0);
    Delay(4000);

}
```

Figure 2-7: Flight Test Code - Arming Speed Controllers

This section is essentially the same as the previous code. We’ve taken it adapted it for multiple motors (each timer/counter can control 2 outputs) and placed it into its own function for ease of use. Things of note are:

1) Ports initialized as outputs are not linearly named, each one had to be looked up
2) The timer/counters were set to have fast PWM, start from the bottom and count up to the max value of 0xFF

In the main code we have 2 things we addressed:

1) Acceleration/deceleration – this was quite tedious as each motor had its own hovering speed so increments needed to be different as well. For the acceleration we needed to get from the motor’s very lowest speed to the hovering speed using delays and increments, we needed 3 different incrementing values. After we hit our hovering speed we essentially did the same process in reverse for the deceleration.

2) Finding each motor’s arming speed and hovering speed – as stated these values were different for each motor so a lot of trial and error went into finding them. Arming speed is simple, choose a value, if it works try one below it until the lowest value is found that still arms the motor.

Hovering speed, however, was difficult and is the reason we ended up tethering each leg individually. Each change in hover speed seemed to throw off the balance of the copter and without gyroscope input we were unable to give it the second by second corrections it needed. Our hovering speeds are very close but still not enough for complete stand alone hovering.

```c
int main (void)
{
    Arm_ESC();

    while (1)
    {
        int i=0, j=0, k=0, z=0;

        for(i=0; i<40; i++){     //Start Motors slow
            if(i%2 == 0){
                j++;            //Increment for M6
                k++;            //Increment for M5
            }
            if(i%4==0){
                k++;            //Increment for M5
            }

            OCR1B = (unsigned char)(143+i);         // Acceleration
            OCR1A = (unsigned char)(143+i);
            OCR0A = (unsigned char)(130+k);
            OCR0B = (unsigned char)(193+j);
            Delay(300);
        }

        OCR1B = (unsigned char)(184);               // Hovering speed
        OCR1A = (unsigned char)(184);
        OCR0A = (unsigned char)(160);
        OCR0B = (unsigned char)(212);
        Delay(20000);                               // Hover for "this" long

        i=0, j=0, k=0;

        for(i=0; i<8; i++){                     // Decelerate for loop
            if(i%2 == 0){
                j++;
                k++;
            }
            if(i%4==0){
                k++;
            }

            OCR1B = (unsigned char)(182-i);         // Decelerate
            OCR1A = (unsigned char)(184-i);
            OCR0A = (unsigned char)(160-k);
            OCR0B = (unsigned char)(212-j);
            Delay(250);
        }

        OCR1B = (unsigned char)(0);             // Stop motors
        OCR1A = (unsigned char)(0);
        OCR0A = (unsigned char)(0);
        OCR0B = (unsigned char)(0);

        return(0);

    }

    return(0);
}
```

Figure 2-8: Test Flight Code

3 Discussion of Difficulties and How They Were Overcome

Over the course of the semester, we encountered numerous problems with various parts of our project in both hardware and software.

Our first objective was to set up a development environment that would allow us to write code in C, convert the C file into a hex file, and then write the hex data onto the microcontroller. Our original setup consisted of using AVR Studio and AVR eXtreme burner, which was recommended by Dr. Elmiligi. The first problem arose when we tried to write our hex file using AVR eXtreme burner. We were able to read and write the original hex file from the board back onto the board, but when we tried a different hex file the burner would cite verification errors. Later we met with an associate of Dr. Elmiligi, and were told to try a different burner offered by AVR called AVRDude, which let us write hex code to our microcontroller through command prompt. Although this method lacked a graphical user interface, it was successful at doing its job, and therefore it was the method we used to upload the hex files onto the microcontroller.

We ran into an issue with the speed controllers after we had successfully been outputting pulse width modulation to the motor ports. Even though there was a waveform going to the motors they wouldn't spin. Due to the speed controllers being ordered from overseas, information on the model and datasheets were not available. This slowed is down significantly, as we were unable to verify the if our code was correct. To solve this problem, we went downtown to our local hobby shop "BC Shaver and Hobbies" to try to get advice from the staff. After describing our problem to them, they tried to control one of the motors using a servo controller; this allowed us to observe the proper arming sequence. Further tests using a function generator let us establish a range of frequencies that would be suitable for the operation of the quadcopter.

For proper operation of the quadcopter, two of the motors that are opposite from each other have to spin clockwise, and the other two have to spin counter clockwise; this is how it will be able to turn and navigate later on in the project. Upon running our code to spin all four motors on the copter, we noticed that the motors were rotating the same way, which did not provide enough lift as 2 motors were generating force against downward. To fix this issue, Dr. Elmiligi rewired and soldered the connections to two of the motors to their respective speed controllers. After the rewiring was completed the motors turned the correct direction and testing could resume.

Near the end of the project we had been changing the code a lot and therefore we had a lot of hex files being sent to the quadcopter. It was frustrating and progress was very slow until we noticed that most of the hex files we thought were changed were identical to the ones used before it. We identified the source of the problem to be AVR studio and the way it saves C and hex files, we were never able to resolve this issue but we adopted a changelog for our C and hex files to better judge whether the file had been changed or not.

Overall, we had difficulties with consistency. Programs that should have worked – didn't, information that should have been easy to find – wasn't, and software that has been tuned and tested by countless others – failed in simple tasks. Consistency again dumbfounded us when each motor needed a different arming value and hovering value outputted to it. There was a good deal more issues with this project than anticipated, but at the end of the semester we learned a great deal more than just how to make a propeller spin.

4 Conclusion

As stated above, we ran into numerous problems with our speed controllers, motors, and software which slowed down the progress of our project quite a bit. Because of these problems, we were not able to make the copter fly autonomously, but we were able to get a great understanding of the concepts behind programming microcontrollers and have positioned ourselves well for the ELEC 499 project. We now have a basis for how pulse width modulation works and have used it to drive motors at many different speeds. At the end of our project we achieved proof of concept by getting enough lift from the propellers that the quadcopter rose more than a foot in the air. All of our work will transfer very easily into a continuation project in ELEC 499.

Finally, we would like to say thank you to our project supervisors, Dr. Fayez Gebali and Dr. Haytham Elmiligi. We have been extremely grateful for their help along the way, as without them this project would not be where it is today. Haytham's constant availability, useful ideas and helpful research were crucial to our project, and we would like to extend our gratitude towards him.

Please find attached our Coding Changelog and Semester Log Book. Final Code is available in Section 6.

5 References

1) *AVR 8-bit Microcontroller with 4/8/16/32K Bytes In-System Programmable Flash Datasheet*, Revision 8161C, Atmel Corporation, May 2009
2) *KK Multicontroller v.5.5 "Blackboard" – The Multicopter Flight Controller,* Rolfe Bakke, Jussi Hermannsen and Mike Barton, www.kkmulticopter.com
3) *Atmega 168 led blink using delay,* Internet: http://tom-itx.dyndns.org:81/~webpage/how_to/atmega168/mega168_led_blink_delay_index.php, Tom L. [November 27th, 2012]
4) *ATmega168A Pulse Width Modulation – PWM,* Internet: http://www.protostack.com/blog/2011/06/atmega168a-pulse-width-modulation-pwm/, ProtoStack June 25th, 2011, [November 17th, 2012]

6 Final Code

The final code used for flight, built to HoverTiedFinal.hex (which can be found on the project website).

```
/*
 * HoverTiedFinal.c
 *
 *  Author: Taylor, Jarrod, Evgeny
 */


#include <avr/io.h>
#include <math.h>

void Delay(int delay);

void Arm_ESC(void);

void Delay(int delay)
{
    int x, y;
    for (x = delay; x != 0; x--)
    {
        for (y = 1000; y != 0; y--)
        {
            asm volatile ("nop"::); //do nothing for a bit
        }
    }
}

void Arm_ESC(void){
    DDRB  = 0b00000110;   // PB1 and PB2 as outputs
    DDRD  = 0b01100000;   // PD5 and PD6 as outputs

    TCCR0A = _BV(COM0A1) | _BV(COM0B1) | _BV(WGM00) | _BV(WGM01);      // Non
inverting mode on OC2A and OC2B, Mode Fast PWM, Bottom Start, 0xFF
    TCCR0B = _BV(CS01) | _BV(CS00);       // Clock/64

    TCCR1A = _BV(COM1A1) | _BV(COM1B1) | _BV(WGM10);      // Non inverting
mode on OC1A and OC1B, Mode Fast PWM 8-bit, Bottom Start, 0x00FF
    TCCR1B = _BV(WGM12) | _BV(CS11) | _BV(CS10);          // Clock/64

    OCR1B = (unsigned char)(0);                          // Zero out all the
outputs
    OCR1A = (unsigned char)(0);
    OCR0A = (unsigned char)(0);
    OCR0B = (unsigned char)(0);
    Delay(4000);
    OCR1B = (unsigned char)(160);                        // Arm motors by
inputing lowest value
```

```
    OCR1A = (unsigned char)(160);
    OCR0A = (unsigned char)(150);
    OCR0B = (unsigned char)(150);
    Delay(4000);
    OCR1B = (unsigned char)(0);                         // Init to zero
before accelerating
    OCR1A = (unsigned char)(0);
    OCR0A = (unsigned char)(0);
    OCR0B = (unsigned char)(0);
    Delay(4000);

}

int main (void)
{
    Arm_ESC();

    while (1)
    {
        int i=0, j=0, k=0, z=0;

        for(i=0; i<40; i++){    //Start Motors slow
            if(i%2 == 0){
                 j++;           //Increment for M6
                 k++;           //Increment for M5
            }
            if(i%4==0){
                k++;            //Increment for M5
            }

            OCR1B = (unsigned char)(143+i);         // Acceleration
            OCR1A = (unsigned char)(143+i);
            OCR0A = (unsigned char)(130+k);
            OCR0B = (unsigned char)(193+j);
            Delay(300);
        }

        OCR1B = (unsigned char)(184);               // Hovering speed
        OCR1A = (unsigned char)(184);
        OCR0A = (unsigned char)(160);
        OCR0B = (unsigned char)(212);
        Delay(20000);                               // Hover for "this" long

        i=0, j=0, k=0;

        for(i=0; i<8; i++){                     // Decelerate for loop
            if(i%2 == 0){
                j++;
                k++;
            }
            if(i%4==0){
                k++;
            }

            OCR1B = (unsigned char)(182-i);         // Decelerate
            OCR1A = (unsigned char)(184-i);
```

```
            OCR0A = (unsigned char)(160-k);
            OCR0B = (unsigned char)(212-j);
            Delay(250);
        }

        OCR1B = (unsigned char)(0);             // Stop motors
        OCR1A = (unsigned char)(0);
        OCR0A = (unsigned char)(0);
        OCR0B = (unsigned char)(0);

        return(0);

    }

    return(0);
}
```

Log book grade (25%): ______________
Report grade (75%): ______________
Total Grade (100%): _______________

Supervisor's Comments:

____

____

____

____

____

____

____

____

____

____

____

____

_________________________ _______________________ ______________________

Supervisor's name (Print) Signature Date

Notes for the supervisor:

1. Please return the marked hard copy to Prof. Tao Lu by Monday, December 10.
2. Attached additional pages for comments if necessary.