OWL ROBOT
User Manual

Microrobot™
www.microrobot.com

CONTENTS

PART1: Owl Robot

1. Introduction

OWL ROBOT is fashioned after an owl and utilizes four RC servomotors. Its CPU is an ATmega8 with an In-System Programming (ISP) function. Thus, the user can download a program to the robot without a ROM Writer. A free C-compiler (Microrobot AVR GCC) is provided. In building this kit the user learns how to control multi RC servomotors. In addition, the free C-compiler and ISP function helps beginners.

2. Features

. ATmega8 (Atmel AVR series, 16MHz X-tal(16 MIPS) but internal 8MHz RC Oscillator setting is required for the RC Servo Source Example. Refer to “Security Bit Settings for ATMega Family.pdf” for the setting.).

. Four R/C servomotors.

. 16 R/C servos connectable (16 I/O port pins).

. Controls up to 8 R/C servomotors at the same time.

. C source code.

. Free Windows C compile (WinAVR AVR GCC).

. Included ISP downloader.

. On board piezo Buzzer.

3. Control

The Owl Robot’s CPU board has sixteen I/O port pins and can control 8 servomotors at the same time. The Owl Robot has an ATmega8 CPU board as a controller. ATmega8 CPU has three internal counters. The CPU board generates up to eight periodic pulses using the timers. The periodic pulses control R/C servomotors.

4. Actions

Six actions are available with the provided sample program.

- Bowing.
- Striding.
- Pecking.
- Nodding.
- Sliding the legs back and forth.
- Skipping.

PART 2 : CPU BOARD

1. Placement Diagram(Silkscreen)

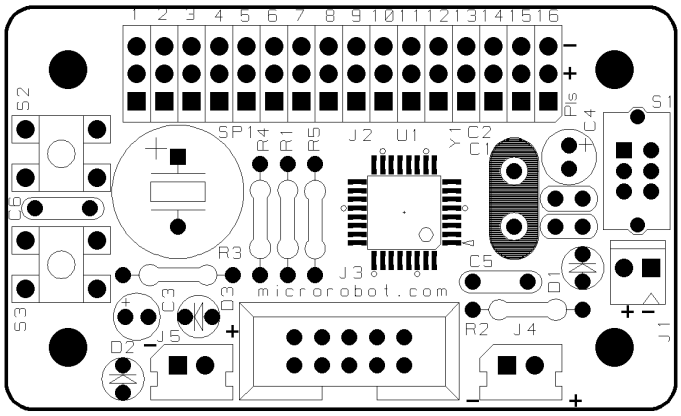
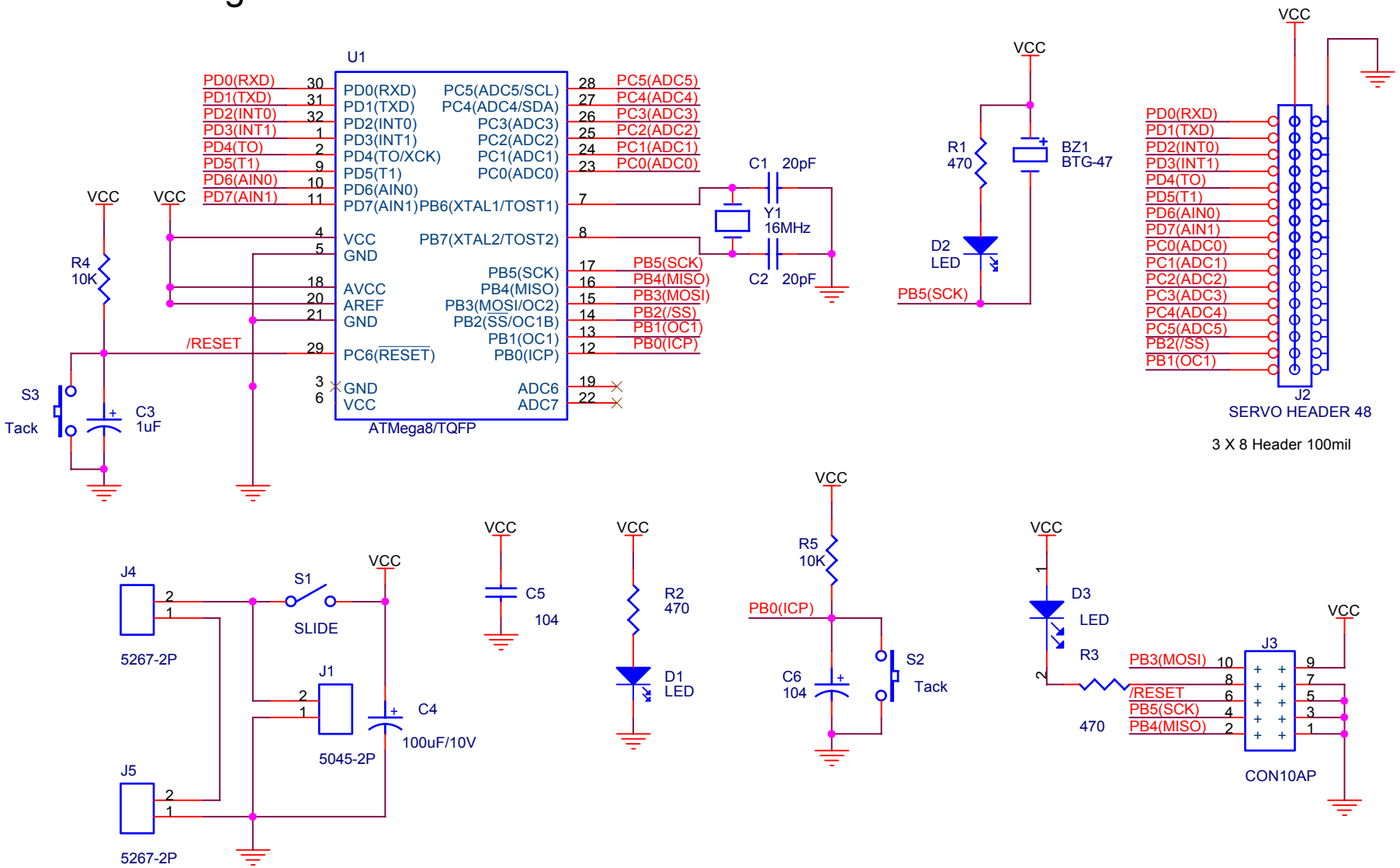


Fig 1.1 ATmega8 Servomotor control board silkscreen.

2. Circuit Diagram



www.microrobot.com		
Title: MR-Servo4433		
Size: A	Document Number: <Doc>	Rev: 1.0
Date:	Sheet 1 of 1	

3. Parts List

NO	Reference	Parts name	Value	Qty.	Remark
1	C1, C2	Capacitor	30pF	2	Ceramic Condenser
2	C3	”	1uF	1	Electrolytic Condenser
3	C4	”	100uF/10V	1	Electrolytic Condenser
4	C5, C6	”	104(0.1uF)	2	Monolithic Condenser
5	D1, D2, D3	LED	RED 3ø	3	
6	J1	Connector	5045	1	5V Power Part
7	J2	”	HEADER PIN(Male)	1	SERVO HEADER 48PIN
8	J3	”	CON10AP	1	HIF3F/10PIN
9	J4, J5	”	5267	2	Battery Power Part
10	R1, R2, R3	Resistor	470Ω	3	
11	R4, R5	”	10K	2	
12	SP1	BUZZER	BTG-47	1	PIEZO
13	S1	S/W	SLIDE S/W	1	
14	S2, S3	”	Tack S/W	2	
15	U1	MCU	ATmega8/TQFP	1	AVR Microcontroller
16	Y1	X-TAL	16MHz	1	ATS type
17		Printed Circuit Board(PCB)		1	Main PCB
18		Servomotor	HES-288	4	
19		Battery Holder & Power Connector	5051-2P	1	AA size * 4
20		Pin head Screw		4	3ø
21		Nut		12	3ø
22		Flat head Screw		4	3ø
23		Downloading Adapter		1	
24		Ribbon Cable		1	1 meter

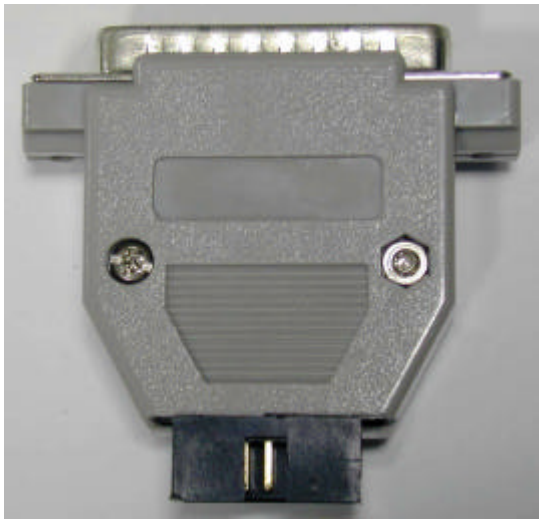


Fig 2.1 Downloading Adapter

Fig 2.2 Ribbon cable

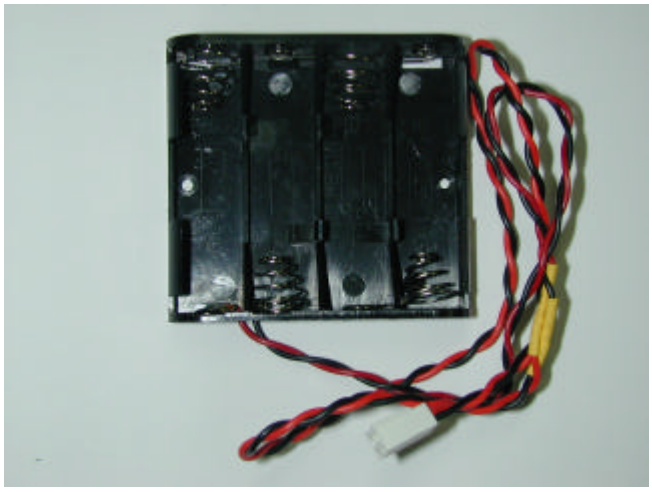


Fig 2.3 Battery Holder & Power Connector

PART 3 : Software Tools

1. AVR Development Program Installation

AVR Development Tools

There are many different kinds of development tools for AVR microcontrollers. Atmel, the AVR CPU manufacturer, provides some AVR development tools free. WinAVR GCC is a free Windows C-compiler.

Wavrasm : AVR assembler, Atmel.
AVR Studio : AVR Emulator/Simulator, Atmel.
AVR ISP : ISP downloading program, Atmel.
PonyProg2000 : ISP downloading program, Lancos. (**Recommended**)
WinAVR GCC : C-compiler, GNU. (**Recommended**)

System requirements for AVR development tools

- Windows 9X/ME or NT/2000/XP
- Pentium-133 or higher
- At least 4 Mbytes of RAM
- CD-ROM Drive

AVR ISP installation:

Run setup.exe in the CD's avr_isp folder.

WinAVR GCC installation

Refer to “How to use WinAVR for Microrobot AVR Products(Eng).pdf”.

2. How to use WinAVR Gcc

Refer to “How to use WinAVR for Microrobot AVR Products(Eng).pdf”.

3. How to use PongProg2000

Refer to the ‘PonyProg Manual for Microrobot AVR Products.pdf’ and the ‘Security Bit Setting for ATMega Family.pdf’ files.

PART 4 : Robot Operation

1. Compile and Download

Compile the Owl Robot source file and download the executable file in the following order.

- Put four batteries into the battery holder and insert the power connector to J1 of the Main PCB.
- Connect the downloading adapter to the PC printer port. Then connect the downloading adapter to the CPU board by using the ribbon cable.
- Turn on the power switch S1 on the control board. LED D1 turns on.
- Download sample code from our website (“How to use WinAVR for Microrobot AVR Products(Eng).pdf”).
- Create a source folder and copy the prototype sample code, including the makefile, from the file you’ve downloaded.
- Type “make all” on the DOS command line platform to compile it.
- Debug and recompile if there are any errors or warnings.
- If there are no errors, the ‘Errors: none’ message appears.
- Run PonyProg2000.
- Do “I/O port setup” properly. Refer to ‘PonyProg Manual for Microrobot AVR Products.pdf’.
- Select ‘Device → AVR micro → ATmega8’.
- Select ‘File → Open Program File’ and load the hex file.
- Select ‘Command → Program’ or press Ctrl + P to start downloading. If a ‘Program Failed’ message appears, select ‘Command → Erase’ or press Ctrl + E to erase the flash memory, and then try to program it again.
- Remove the ribbon cable from the CPU board and restart the board.

2. R/C servomotor(HES-288)

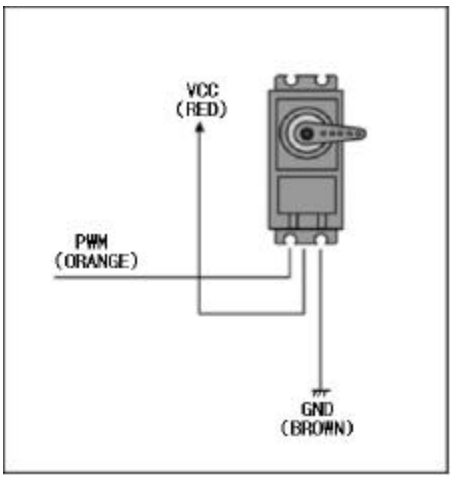
An R/C servomotor is a special type of geared motor. It has its own internal controller, which uses feedback to control the DC motor according to the PWM (Pulse Width Modulation) position signal. The degree of turn of the servomotor's axle depends upon the input pulse width signal. Generally, it turns from 0° to 180°(or -90°~+90°). Therefore the servomotor does not need an encoder to get the axle's current position.

HES-288 is used in this Robot. It is controlled using three wires, as is general servomotor. Two wires are for the power input and the other wire is for the PWM control signal input.

HES-288 Specification

- Torque : 2.5 kg/cm
- Speed : 60 degrees/0.2sec
- Size : 41 x 20 x 35 mm (length x depth x height)
- Weight : 42g
- Voltage : 4 ~ 6 V
- Operating Frequency : 50Hz~100Hz (Period:10msec~20msec)

Wire connection



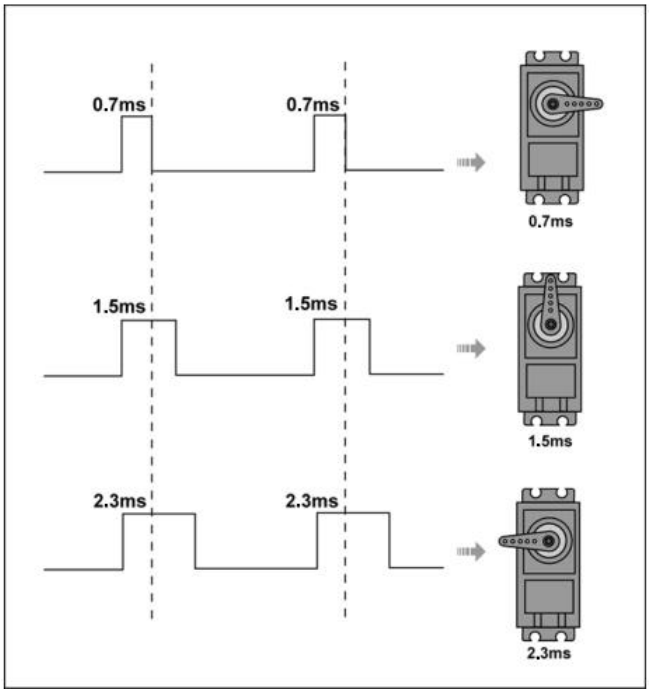
Red: Vcc(DC 4~6V)

Brown: Gnd

Orange: PWM control signal input.

Fig 4.1

Control signal



0.7 msec : + 90°

1.5 msec : 0°(Center)

2.3 msec : - 90°

To keep the desired position (or holding torque), the pulses must be provided every 10 ms to 20 ms.

Fig 4.2

3. Adjusting motors

When the power is on, the servos, which are connected to the board, should turn to the center but they might not. This is because the characteristics of the individual servomotors are a little different. To adjust each motor, the initial setting values in the source code must be changed.

- The ATmega8 CPU board can control eight servomotors at the same time. The eight servomotors are named SERVO_0, SERVO_1,… SERVO_7 in the program source.
- On the upper part of the PCB, index numbers from 1 to 16 are marked for the servomotor ports. Refer to Fig 1.1 above. Port numbers 1 to 8 correspond to SERVO_0 to SERVO_7 respectively. OWL Robot uses the following four servomotors.

  SERVO_0: Left arm.

  SERVO_2: Right arm.

  SERVO_4: Left leg.

  SERVO_6: Right leg.

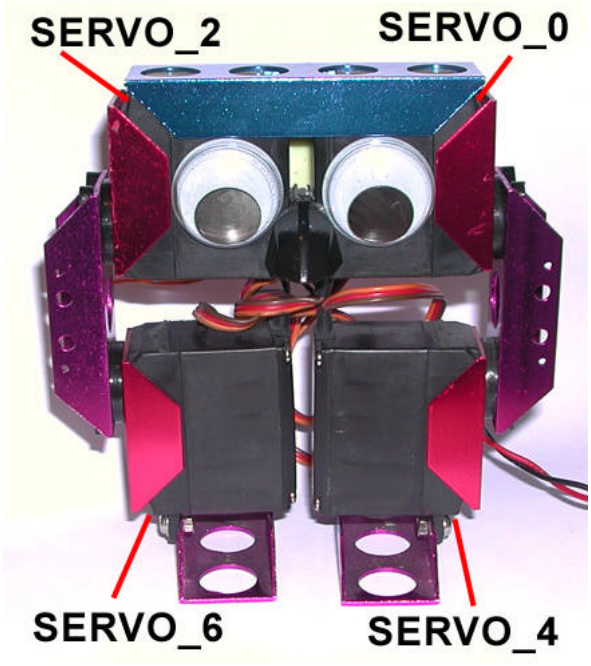


Fig 4.3 Servo Motor Connections

- Run your editor and open the file **owl.c**.
- At the beginning of the source code, there is some user definition codes.

```
/******************* User definition ***************************/

#define SERVO_0          (1500*8)

#define SERVO_1          (1550*8)

#define SERVO_2          (1450*8)

#define SERVO_3          (1400*8)

#define SERVO_4          (1350*8)

#define SERVO_5          (1550*8)
```

#define SERVO_6 (1480*8)

#define SERVO_7 (1500*8)

Note : The actual values might differ from those above.

The defined values are initial center (pulse) values for each servomotor. The center value refers to a time constant which makes a servomotor's axle turn to the center (0°). Refer to Fig 4.2. The user must determine these center values by actually testing their servomotors. In the above definition, (1500*8) means 1.5 ms, which is a typical center value. When the value is changed by ±1*8, the axle turns approximately ±0.1125°. Repeat the process of source modification, compiling and downloading to the robot to determine the final correct center values.

For example, when one servomotor's axle turns 5 degrees counter clockwise past the center position, subtract 44(=5/0.1125) from the initial value.

- Below are some other definitions in the source file.

  #define FACTOR_0 1024

  #define FACTOR_1 1024

  #define FACTOR_2 1165

  #define FACTOR_3 1024

  #define FACTOR_4 960

```
#define FACTOR_5          1024
#define FACTOR_6          980
#define FACTOR_7          1024
```

Note: The actual values might differ from the above.

The above values (constants) are used as turning factors. To make the servos turn properly, the above values should be adjusted along with a testing process.

There is a special test mode to determine the above constants. Press and release the S3(Reset) key while pressing the S2 key and then release the S2 key. Each servomotor should turn 90°. If not, change the corresponding FACTOR values in the source code. For example, if the SERVO_0 turns more than 90 degrees, reduce FACTOR_0. If the SERVO_2 turns less than 90 degrees, increase FACTOR_2.

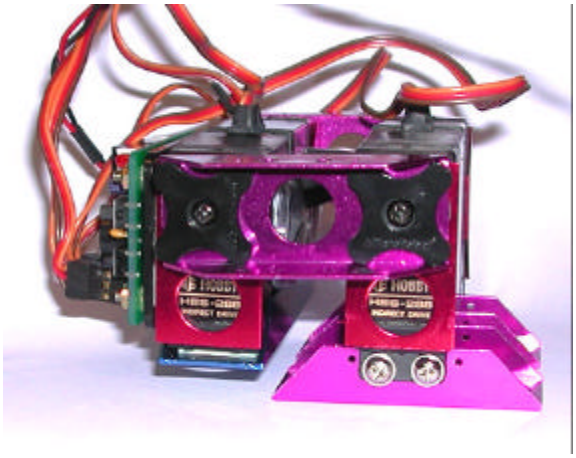


Fig 4.4 Servo Motor Angle Settings

- After determining the proper values, press the S3 (Reset) key to cause the robot to stand straight. Press the S2 key to start the robot working.

PART 5 : Source Codes

Be aware that the following source code assumes that the CPU crystal is 8MHz.

- owl.c : Main program.
- io.h : Header file which has AVR CPU's registers and I/O pin definitions.
- signal.h : Interrupt vectors functions and interrupts vector table definitions.
- init.c : Basic initialization.
- function.c : Delay, buzzer and switch functions.
- motion.c : Servomotor control functions.
- inter.c : Interrupt routine functions.

1. Init.c

```
void port_init(void)
{
      outp( 0xfe, DDRB );     //LSB of the port B = input pin, the other pins = output pin.
      outp( 0xff, DDRC );     //Set all PORTC pins to output.
      outp( 0xff, DDRD );     //Set all PORTD pins to output.

      outp( 1, PORTB );       //Set the bit0 of the Port B to use the internal full-up resistor.
      outp( 0, PORTC );
      outp( 0, PORTD );
}
```

Initializes ports.

```
void motor_init(void)
{
      outp(4, TCCR0);                  // TIMER_0 CK/256 -> 1=32usec

      outp(0, TCCR1A);
      outp(1, TCCR1B);                 // CTC1 clear, TIMER_1 CK/1 -> 8=1usec

      outw(OCR1A, 0xffff);;  // 65535 -> (65535/8)usec = 8.192msec
}
```

Initializes two timers to generate continuous pulses with a constant period (20 ms), which are used to control the servomotors.

MCU main clock is 8 MHz (1/8 us). The timer0's prescaler is set to 256. This means that the timer0 clock is 32 us (1/8 us * 256). Timer1 uses the main clock (1/8 us) without prescaling.

```
void motor_enable(void)
{
	outp(178,TCNT0);	// 256-178=78 -> 78*32usec=2.496msec
	outw(TCNT1L, 0);

	cbi(TIFR, TOV0);	// clear TOV0 : Timer0 Overflow Flag clear
	sbi(TIMSK, TOIE0);	// set TOIE0 : Timer0 overflow Interrupt enable

	cbi(TIFR, OCF1A);	// clear OCIF1 : Timer1 Output Compare A Mach Flag clear
	sbi(TIMSK, OCIE1A);// set OCIE1A : Timer1 Output Compare A Mach Interrupt Enable
}
```

Sets time-constant to the timer0 and enables timer0 interrupt. Even if the timer0 interrupt is enabled, the timer0 interrupt will not occur until the global interrupt bit is enabled.

```
void motor_disable(void)
{
	outw(TCNT1L, 0);
	outp(0, TCNT0);
	outp( 0, PORTD );
	cbi(TIMSK, OCIE1A);// clear OCIE1A : Timer1 Output Compare Interrupt disable
	cbi(TIMSK, TOIE0);	// clear TOIE0 : Timer0 overflow Interrupt disable
}
```

Function to disable the servomotors.

```
void motor_zero(void)
{
	int i;

	Init(0)=SERVO_0;
	Init(1)=SERVO_1;
	Init(2)=SERVO_2;
	Init(3)=SERVO_3;
	Init(4)=SERVO_4;
	Init(5)=SERVO_5;
	Init(6)=SERVO_6;
	Init(7)=SERVO_7;

	Factor(0)=FACTOR_0;
	Factor(1)=FACTOR_1;
	Factor(2)=FACTOR_2;
	Factor(3)=FACTOR_3;
	Factor(4)=FACTOR_4;
	Factor(5)=FACTOR_5;
	Factor(6)=FACTOR_6;
	Factor(7)=FACTOR_7;

	for(i=0; i<8; i++)
	{
		Deg(i)=Speed(i)=0;
		Pulse(i)=Init(i);
	}
}
```

Copies user adjusted (defined) constants to the variables. The constants can be used directly instead of being copied to the variables. However, for the upgrade version the variables are used. In the upgrade version, the variables may be also filled out with constants from the PC.

```
void system_init(void)
{
	port_init();
	motor_init();
	motor_zero();
}
```

System initialization.

2. Function.c

```
void delay(int ticks)
{
	while(ticks--);
}

// 1msec UNIT delay function
void delay_1ms(unsigned int i)
{
	word j;
	while(i--)
	{
		j=2000;   // 8Mhz
		while(j--);
	}
}
```

1 ms based delay function. For example, to make a 1 sec time delay, call delay_1ms(1000). This function does not consider other interrupts. This means that other interrupts during the delay function can make the delay time longer.

```
void buzzer(void)
{
	word i;
	for(i=0; i<50; i++)
	{
		BUZZER_ON;
		delay(BUZ_DLY1);
		BUZZER_OFF;
		delay(BUZ_DLY1);
	}

	for(i=0; i<50; i++)
	{
		BUZZER_ON;
		delay(BUZ_DLY0);
		BUZZER_OFF;
		delay(BUZ_DLY0);
	}
}
```

Makes a buzzing sound using the BTG-47 buzzer. The BTG-47 is frequency (pulse) activated.

```
void until_switch_intput(void)
{
	loop_until_bit_is_clear(PINB, 0);
	loop_until_bit_is_set(PINB, 0);
}
```

Push button switch S2 input function, which waits until the S2 key is pressed and released.

3. Inter.c

In the source, two different interrupts are used. The interrupts are exclusive. This means, when one interrupt service routine is served, the other one must wait.

```
SIGNAL(SIG_OVERFLOW0)
{
	outw(TCNT1L, 0);
	outp(178,TCNT0);			// 256-178=78 -> 78*32usec=2.496msec

	sbi(PORTD, No);

	if( Pulse(No) > (Init(No)+Deg(No)*71+(Speed(No)/2)) )	Pulse(No)-=Speed(No);
	else if( Pulse(No) < (Init(No)+Deg(No)*71-(Speed(No)/2)) )		Pulse(No)+=Speed(No);
	else  Flag_motor&=~(1<<No);

	outw(OCR1A, Pulse(No));
	sbi(TIMSK, OCIE1A);  // set OCIE1 : Timer1 Output Compare Interrupt enable

	No++;
	No&=0x07;

	cbi(TIFR, TOV0);			// clear OCIE1 : Output Compare Flag1 clear
}

SIGNAL(SIG_OUTPUT_COMPARE1A)
{
	outp(0, PORTD);			//clear Port D
	cbi(TIMSK, OCIE1A);  // clear OCIE1 : Timer1 Output Compare Interrupt disable
}
```

Timer0 overflow interrupt occurs every 2.5 ms. In the interrupt service routine, one of eight-servomotor control pins is activated by turns. This means each servomotor has a controlled-focus every 20 ms(2.5msec*8).

'No' is an index variable that indicates the port number to be serviced. The 'No' value can be set from 0 to 7. The corresponding port pin is set high and OCR1A is set to the high-pulse time variable Pulse(No). After

the set time has elapsed, the output_compare1a interrupt occurs, which makes the pulse low.

The Speed(No) value is used to control the servomotor turning speed. For example, when the Speed(No) is set to 71, the servo turns 1°/20ms(=50°/sec).

It takes time for a servomotor to turn to its destination. The Flag_motor variable is used to check whether the servomotors are finished turning. Flag_motor is a one-byte variable. The bit value indicates the servomotor's status. 1 means the motor is still turning and 0 means the turning is done.

The servomotor turns 1° when the pulse width is changed by 8.888 us. The timer1's clock is 1/8 us. Thus the servomotor turns 1° per 71 timer1 clock ticks.

Init(No) is the time constant that sets the servomotor's axle at the center(0°) position.

4. Motion.c

```
void wait_until_move(void)
{
	Flag_motor=0xff;		// Initialze with 0xff( all servos are still turning.)
	while(Flag_motor);		// Wait until all servomotor finish turning.
}
```

Waits until all servomotors have finished turning before starting the new next action.

```
// Turn the servo to the absolute angle at the speed.
void absolute_deg(int no, int deg, int speed)
{
	Deg(no)=(int)((long)(deg)*(long)(Factor(no))>>10);
	Speed(no)=(int)((long)(speed)*(long)(Factor(no))>>10);
}
```

Turns the servo(no) to the absolute angle(deg) at the given speed(speed).

The shift operation(>>) is used instead of a divide operation to increase the operation speed.

The Factor(no) is used to calculate the Speed(no) as well as the Deg(no). This is because each individual motor has different characteristics.

```
void same_absolute_deg(byte flag, byte dir, int deg, int speed)
{
	int i;

	wait_until_move();
	for(i=0; i<8; i++)
	{
		if( flag&(1<<i) )
		{
			if(dir&(1<<i))  absolute_deg(i, deg, speed);
			else			absolute_deg(i, -deg, speed);
		}
	}
}
```

Moves the motors(up to eight motors) at the same time, to the same absolute angle. The angle(deg) and

the speed are applied to all enabled servomotors equally, but the direction can be selected with the dir parameter : clockwise or counterclockwise.

<Parameters>

flag : Motor Enable flags. Each bit enables the corresponding servomotor. 1 = enable, 0 = disable.

dir : Direction flags for each motor. 1= use +deg, 0 = use -deg.

deg : Absolute angle to turn.

speed : Turning speed.

This function is used when several motors need to move to the same angle or symmetric angle.

For example,

same_absolute_deg(S0|S2|S4|S6, S2|S4, 45, speed);

-Enables motors 0,2,4 and 6.

-Motors 1 and 3 turn to the –45 degree position and motors 0 and 4 turn to the 45 degree position at the set speed.

same_absolute_deg(S0|S2, S2, 15, speed);

-Enables motors 0 and 2.

-Motor 0 turns to the –15 degree position and motor 2 turns to the 15 degree position at the set speed.

```
// relative angle turning function for the each motor.
void relative_deg(int no, int deg, int speed)
{
	int temp;
	temp=(int)(((long)(Deg(no))<<10)/(long)(Factor(no)));
	deg+=temp;
	Deg(no)=(int)((long)(deg)*(long)(Factor(no))>>10);
	Speed(no)=(int)((long)(speed)*(long)(Factor(no))>>10);
}
```

Turns motors to the relative angle at the given speed. The **relative angle** means the angle relative to the current position (as opposed to the start position).

```
void same_relative_deg(byte flag, byte dir, int deg, int speed)
{
	int i;

	wait_until_move();
	for(i=0; i<8; i++)
	{
		if(flag&(1<<i))
		{
			if(dir&(1<<i))  relative_deg(i, deg, speed);
			else			relative_deg(i, -deg, speed);
		}
	}
}
```

This function can be used when several motors need to move to the same relative +angle or -angle. This is the same function as the “same_absolute_deg()” except that the deg is a relative angle.

bow_motion() : Bowing.

walking_motion() : Striding.

neck_motion() : Pecking.

neck1_motion() : Nodding.

run1_motion() : Sliding the legs back and forth (Shuffling).

run2_motion() : Skipping.

5. Owl.c

```
void adjust_mode(void)
{
	until_switch_intput();
	same_absolute_deg(S4|S6, S4, 90, 100);
	same_absolute_deg(S0|S2, S2, 90, 100);
	buzzer();                                    //Buzzing sound
}
```

Special test mode function to adjust all servomotors. Press and release the S3 (Reset) key while pressing the S2 key, and then release the S2 key to enter this mode. Each servomotor should turn 90°.

```
int main(void)
{
	system_init();				//System Initialize
	buzzer();				//Buzzer
	motor_enable();

	sei();					// Global interrupt enable

	if(bit_is_clear(PINB,0))	adjust_mode();

	until_switch_intput();
	delay_1ms(500);

	buzzer();

	delay_1ms(2000);

	while(1)
	{
		bow_motion(27);			// Bowing
		delay_1ms(1000);

		walking_motion(27);		// Striding
		delay_1ms(1000);

		neck1_motion(400);		// Pecking

		delay_1ms(1000);

		neck2_motion(100);		// Nodding
		neck2_motion(100);
		delay_1ms(1000);
		run1_motion(100);		// Sliding the legs back and forth
		run1_motion(100);
		delay_1ms(1000);

		run2_motion(150);		// Skipping
		delay_1ms(1000);
	}
}
```

Main function: Initialization, interrupt enable and robot action functions.