APPENDIX A

Kyubot Written Report

Source Code

```
/**     UF EEL 5666 Inteligent Machine Design Lab Course Project Source Code
**
**      Date Created: March 13, 2008
**      Author : Kyuhyong You       MS in Mechanical Engineering
**      E-mail : kyuhyong@ufl.edu or kyuhyong@gmail.com
**
**      <Description>
**      This program start with reading the waypoint data stored in SD-com Module.
**      The data includes number of waypoint data which the robot will fallow.
#include <hidef.h>                      /* common defines and macros */
#include <mc9s12dp256.h>    /* derivative information */
#include <math.h>
#include <float.h>
#include <stdio.h>
#include <stdlib.h>

#include "my_vectors.h"

/*************************************
* COMPILE OPTIONS
*************************************/
#define    PID_TEST_MODE0                   //When checking PID controller
#define SONAR_TEST_MODE         0               //Not yet functional

#define SDCOM_TEST_MODE         0
#define NEW_NMEA_MODE           1               //Use NMEA parsing method (Must be 1)

#define PID_RECORD_MODE         0               //Save PID Data into the SD memory
#define GPS_RECORD_MODE         0               //Save GPS tracking data into the SD memory

#define ON_OFF_CONTROL_MODE             1               //Use On-Off Control Mode : So far most stable Controller
#define PID_CONTROL_MODE                0               //Use PID Control Mode : Faster yet unstable control

#define SERVO_PERIOD                    20000                   // Set servo period as 20ms
#define SERVO_CENTER_ANGLE              1550
/*************************************
* LCD Command
*************************************/
#define LCD_ENABLE (PORTA_BIT2 = 1)
#define LCD_DISABLE (PORTA_BIT2 = 0)
#define    LCD_CLEAR                0x01
#define CURSOR_HOME             0x02
/*****************************************
* LCD Location
*****************************************/
#define line_1      0x80
#define line_2              0xC0
#define line_3              0x94
#define line_4              0xD4
/*****************************************
* MOTOR Control Options
*****************************************/
```

```
#define MOTOR_ENABLE (PORTB_BIT4 = 1)
#define MOTOR_DISABLE (PORTB_BIT4 = 0)
#define GoForward (PORTB_BIT3 = 0)
#define GoBackward (PORTB_BIT3 = 1)
#define    PI                    3.141592
#define SATURATION_L     20
#define SATURATION_R     15
static long absoluteTime = 0;
struct {
        unsigned int state;
        int ptr;
        volatile unsigned char Temp[100];
        unsigned char NS;
        unsigned char EW;
        volatile unsigned int GPS_Received;
        unsigned int rcvCmpl;
        unsigned int d_Lat[2];
        unsigned int d_Lon[2];
        unsigned int Current_HDG;
        unsigned int Current_Speed;
}gpsData;
struct{
        unsigned int Distance[100];
        int Maneuver;
        int Decellerate;
}Obstacle;

volatile unsigned int ANGLE_TIME = SERVO_CENTER_ANGLE;
unsigned char temp;
unsigned char sci1_temp;
unsigned int checkSCI0;
unsigned int checkSCI1;
/***********************************************
Sonar Control Variables
***********************************************/
unsigned int Sonar_Center = 37;
volatile unsigned int Sonar_L, Sonar_R;
//int Maneuver;
volatile unsigned int CURRENT_ANGLE;

volatile unsigned int Encoder_Count;
volatile unsigned int Rotary_Input;
volatile unsigned int Encoder_time;

int motorDty = 0;
unsigned int Encoder_Display_Enable;
unsigned int PID_Enable =0;

int ServoTemp;
unsigned int Goal_RPM;
unsigned int Old_RPM_Error1, Old_RPM_Error2;
unsigned int Old_Time;

long Current_Position=0,Bauckup_Positon=0, Old_Position=0,Old_Position2=0, Goal_Position = 0;
long Position_Error=0,D_Position_Error=0,I_Position_Error=0,PID_Position=0,D_Error_Backup;

unsigned int checkButton1 = 0;
unsigned int checkButton2 = 0;
```

```
unsigned int SDRead = 0;
/************************************************8
* PID CONTROL GAIN    (To be implemented)
************************************************/
float P_Gain        =       0;
float I_Gain        =       0;
float D_Gain        =       0;

/************************************************8
* ATD CONVERTING VARIABLES
************************************************/
volatile unsigned int P_Tuner;
volatile unsigned int D_Tuner;
volatile unsigned int Accel_X;
volatile unsigned int Accel_Y;
volatile unsigned int Accel_Z;
volatile unsigned int SONAR_1;
volatile unsigned int SONAR_2;

/************************************************
* Operation Environment SET UP
************************************************/
unsigned int onGPS_record = 0;
unsigned int onAvoidance = 0;
unsigned int onPID_Control = 0;

struct{
        unsigned int ptr;
        unsigned int n_Points;                  //Number of Waypoints
        unsigned int state;                     //Waypoint Parsing state
        unsigned int readOk;                    //Indicate Reading Complete
        unsigned int processOk;
        unsigned int length;
        volatile unsigned char Data[250];
        unsigned char Temporary[10];
        //unsigned int d_wayPoint[10][4];       //Latitude [ddmm][ss], Longitude[ddmm][ss]
        volatile unsigned int i_Lat[100][2];
        volatile unsigned int i_Lon[100][2];
}wayPoint;

struct{
        unsigned int Route;             //Current path route 1, 2, 3 ...
        unsigned int state;             //Current Navigation status 0: normal, 1: obstacle avoidance
        unsigned int Target_Coordinate[4];
        //float      Goal_Heading;
        unsigned int Last_Heading;
        unsigned int Last_Coordinate[4];        //Lastly known coordinate
}Navi;
void SCI0_Init(void);
void SCI1_Init(void);
void sendByteSCI0(unsigned char text);
void sendByteSCI1(unsigned char text);
void sendMsgSCI0(unsigned char *text);
void sendMsgSCI1(unsigned char *text);
void delay_ms(unsigned int m);
void wait_busy_flag(void);
void delay_us(unsigned int p);
void instruction_out(unsigned char b);
void char_out(unsigned char b);
```

```
void string_out(unsigned char b, unsigned char *str);
void wait_SDcom(void);
void Process_Waypoint(void);
void Parse_Waypoint(int i);
void PID_Control(int Current_RPM);
void Parse_GPS(void);
void display(unsigned char ln,int t1, int t2);
#pragma CODE_SEG __NEAR_SEG NON_BANKED /* Interrupt section for this module. Placement will be in NON_BANKED area. */

interrupt void RTI_ISR(void) { /* simple RTI interrupt service routine */
   absoluteTime++;
   /* clear RTIF bit */
   CRGFLG = 0x80;
}

/************************************************************************
 SCI0_ISR
 - SCI channel 0 interrupt service routine
************************************************************************/
interrupt void SCI0_ISR(void) { /* SCI interrupt service routine */
        int i;
        unsigned char comp[5];
        temp = SCI0SR1; //dummy read(clear flag)
        temp = SCI0DRL; //read data
        if(gpsData.GPS_Received == 1) return;
        //DisableInterrupts;
        switch(gpsData.state){
                case 0:
                        if(temp == 0x24){                       //wait for receiving '$'
                                PORTB_BIT6 = PORTB_BIT6^1;
                                gpsData.ptr = 0;
                                gpsData.state = 1;
                        }
                        break;
                case 1:
                        gpsData.Temp[gpsData.ptr++]=temp;
                        if(temp == 0x2a){                       // IF '*' is found wait for another NMEA
                                for(i=0;i<5;i++){
                                        comp[i]=gpsData.Temp[i];    // Put first 5 character into comp

                                }
                        if(strncmp(comp,"GPRMC",5)==0)     {                   //      Compare      if "GPGGA" received
                                PORTB_BIT5 = PORTB_BIT5^1;
                                gpsData.GPS_Received =1;
                        } else gpsData.GPS_Received = 0;
                                gpsData.state = 0;
                        }
                        break;
        }
        //EnableInterrupts;


}
/************************************************************************
 SCI1_ISR
 - This interrupt is set when DS module send data
************************************************************************/
```

```
interrupt void SCI1_ISR(void) { /* SCI interrupt service routine */
	sci1_temp = SCI1SR1; //dummy read(clear flag)
	sci1_temp = SCI1DRL; //read data
	//PORTB_BIT4 = PORTB_BIT4^1;
	switch(SDRead){
		case 0:
			break;
		case 1:
			wayPoint.Data[wayPoint.ptr++]=sci1_temp;
			if(sci1_temp=='#') {
				wayPoint.readOk = 1;
				wayPoint.length = wayPoint.ptr;
			}
			break;
	}
}

interrupt void Timer_Compare2_ISR(void){
	unsigned int time = 0;
	/*if( PTP & 0x01){
		time = SERVO_PERIOD - ANGLE_TIME;
	    }
	else {
		time = ANGLE_TIME;
	}*/
	ServoTemp = SERVO_PERIOD - ( ANGLE_TIME << 1);
	time = (ServoTemp * ( PTP & 0x01 )) + ANGLE_TIME;
	PTP= PTP ^ 0x01;
	TC2 = time + TCNT;
	TFLG1 |= 0x04;
}
interrupt void Timer_Compare3_ISR(void){
	int Current_RPM;
	unsigned char send1[20];
	DisableInterrupts;
	TC3 = TCNT + 25000;                              // Every 50ms = 25000 / 500
	PORTB_BIT7 = PORTB_BIT7^1;
	Encoder_Count = PACN32;
	Current_RPM = Encoder_Count;
#if PID_TEST_MODE
		sprintf(send1,"E:%4d|RPM:%4d|",Encoder_Count,Current_RPM);
		if(Encoder_Display_Enable) string_out(line_4,send1);
#endif
	if(PID_Enable ==1) PID_Control(Current_RPM);
	Encoder_Count=0;
	PACN32=0;
	TFLG1 |= 0x08;
	EnableInterrupts;
}
interrupt void Timer_Compare4_ISR(void){
	unsigned int Sonar_Value;
	PORTB_BIT1=PORTB_BIT1^1;
	TC4 = TCNT + 55000;
	TFLG1 |= 0x10;
}
/************************************************
* ATD CONVERTING INTERRUPTS
************************************************/
interrupt void ATD0_ISR(void){
```

```
	Rotary_Input= ATD0DR2;
	P_Tuner = ATD0DR3;
	D_Tuner = ATD0DR4;
	Accel_X = ATD0DR5;
	Accel_Y = ATD0DR6;
	Accel_Z = ATD0DR7;
}
interrupt void ATD1_ISR(void){
	SONAR_1 = ATD1DR6;
	SONAR_2 = ATD1DR7;
}
/************************************************
* INPUT BUTTON CAPTURE INTERRUPTS
************************************************/
interrupt void Input_Capture0_ISR(void){
	checkButton1 = 1;  TFLG1 |= 0x01;}
interrupt void Input_Capture1_ISR(void){
	checkButton2 = 1;
	TFLG1 |= 0x02;
}
#pragma CODE_SEG DEFAULT
void RTIInit(void) {
   /* setup of the RTI interrupt frequency */
   /* adjusted to get 1 millisecond (1.024 ms) with 16 MHz oscillator */
   RTICTL = 0x1F; /* set RTI prescaler 16*2^10 */
   CRGINT = 0x80; /* enable RTI interrupts */
}
void SCI1_Init(void){		/* Serial port initialization */
	SCI1BDL = 52;			// For SD-com controller communication
	SCI1CR2=(1<<2)|(1<<3)|(1<<5);		// Enable transmitter inteerupt
}
void SCI0_Init(void){		/* Serial port initialization */
	SCI0BDL = 104;			/* set Baud rate as fallowing @8Mhz */
	SCI0CR2=(1<<2)|(1<<3)|(1<<5); //riceive interrupt enable
	gpsData.rcvCmpl =0;
}
void LCD_Init(void){
	DDRA = 0xFF;	// Set PORTA as OUTPUT
	PORTA = 0xFF;	// Clear PORTA
	LCD_ENABLE;					// LCD ENABLE
	delay_ms(15);
	delay_us(600);
	instruction_out(0x28);		//LCD FUNCTION SET (16x2 LINE, 4BIT, 5x8 DOT)
	delay_ms(2);
	instruction_out(0x28);		//LCD FUNCTION SET (16x2 LINE, 4BIT, 5x8 DOT
	delay_ms(2);
	instruction_out(0x0C);		//LCD DISPLAY ON, CURSOR OFF, BLINK OFF
	instruction_out(0x06);		//LCD ENTRY MODE SET
	instruction_out(0x02);		//RETURN HOME
	instruction_out(0x01);		//LCD CLEAR
	instruction_out(0x01);		//LCD CLEAR
	delay_ms(4);
}
void PORT_Init(void){
	DDRB = 0xFF;
	PORTB = 0x00;
	PTP =	0x00;
	DDRT =	0x00;			// set PORTT as input : PT7 as Pulse Accumulator input
	DDRP = 0x01;			// Set PWM PORT : PP0 as PWM output for servo Period
```

```
        DDRH = 0x00;                // Set PORTH as input
        PERH |=0b11100000;          // PORTH channel 5,6,7 Pull up
        PPSH = 0x00;                // PORTH Polarity: Falling Edge
        DDRJ = 0xFF;
        PTJ = 0x00;
}
void Timer_Init(void) {
        TIOS =   0b00011100;                //        Output Compare Select
        CFORC = 0b00000100;                 //        Force
        OC7M =  0b00000100;                 //        Mask
        OC7D =  0b00000100;                 //        Data
        TSCR1_TEN =     1;                          //        TCNT Enable
        TIE =    0b00011111;                //        Timer Interrupt Enable
        TSCR2 = 0x03;                       //        Prescale 8
        TCTL4 = 0b00001010;                 //         Input Capture 1,0 Falling Edge
    PACTL_PAEN = 1;                         // Enable Pulse Accumulator
    PACTL_PEDGE = 1;            // Rising edge
                                // 0x40 : 0b01000000
        PACN32 =        0;                  // Pulse Accumlator Counter 2,3 initialize
        TC2 = 0xffff;                       // Timer Compare Interrupt 2
        TC3 = 0xffff;                       // Timer Compare Interrupt 3
        TC4 = 0xffff;                       // Timer Compare Interrupt 4
}
void ATD_Init(void){
        ATD0CTL2 = 0b11000010;  // Set ADPU, Set AFFC, Set ASCIE
        ATD1CTL2 = 0b11000010;
        ATD0CTL3 = 0x00;                                        // Select active background mode
        ATD1CTL3 = 0x00;
        ATD0CTL4 = 0b00000111;  //Set 10-bit,   SET PRS2,PRS1,PRS0 total division factor is 9x2=18
        ATD1CTL4 = 0b00000111;
        ATD0CTL5 = 0b10110000;  // Set Right justified, Set SCAN Mode, MULTI
        ATD1CTL5 = 0b10110000;
        ATD0DIEN = 0x00;                // Disable digital input buffer to PTADx
        ATD1DIEN = 0x00;
}
void PWM_Init(void) {
        PWME_PWME2 = 1;                 //Enable pwm channel 2
        PWMPOL_PPOL2 = 1;       //SET POLARITY
        PWMCLK_PCLK2 = 0;       //Select Clock B for channel 2
        PWMCAE_CAE2 = 0;        //Channel 3 operates in Left Aligned output mode
        PWMPRCLK =   0b00000111;                //Clock A = bus clock / 2
        PWMPRCLK |= 0b00010000;                 //Clock B = bus clock / 2
        DDRP =0xFF;
        PWMPER2 = 0x80;                 // Left Aligned : PWM Period = Channel clock period
        PWMDTY2 =       0;              // 0<DTY<128
        PWMCNT7 = 0;
        PWME_PWME7 = 1;                 //Enable PWM Channel 7
        PWMPOL_PPOL7 = 1;       //SET POLARITY
        PWMCLK_PCLK7 = 1;       //Select Clock SB for channel 2
        PWMSCLB = 80;
        PWMPER7 = 0xFF;
        PWMDTY7 = 37;   // 15<DTY<60
        PWME_PWME6 = 1;
        PWMPOL_PPOL6 = 1;       //SET POLARITY
        PWMCLK_PCLK6 = 1;
        PWMPER6 = 0xFF;
        PWMDTY6 = 0;
}
void startTimeBase(void){
```

```
   absoluteTime = 0;
   RTIInit();
   EnableInterrupts;
}
#pragma CODE_SEG DEFAULT

/*****************************************************************************
  sendByteSCI0, sendMsgSCI0, sendInt
  - send any char.s or messages to PC
*****************************************************************************/
void sendByteSCI0(unsigned char text){
        while (!(SCI0SR1 & 0x80));    /* wait for output buffer empty */
        SCI0DRL = text;
}
void sendMsgSCI0(unsigned char *text){
        while (*text != '\0') {
                sendByteSCI0(*text);
                text++;    }
}
void sendByteSCI1(unsigned char text){
        while (!(SCI1SR1 & 0x80)); /* wait for output buffer empty */
        SCI1DRL = text;
}
void sendMsgSCI1(unsigned char *text){
        while (*text != '\0') {
                sendByteSCI1(*text);
                text++;
        }
        sendByteSCI1(0x0D);sendByteSCI1(0x0A);                  //For SDcom module command
}

void wait(long ms){
   /* definition is 1 ms */
   long timeout;
   timeout = absoluteTime + ms;
   while (timeout != absoluteTime) {
      __asm NOP;
      /* __asm WAI; */ /* will be waken up by the RTI exception. Not well supported in BDM mode */
   }
}

void wait_busy_flag(void){
        unsigned int i;
        for(i=0;i<10000;i++);
}
void delay_ms(unsigned int m){
        unsigned int i, j;
        for(i=0;i<m;i++){
                for(j=0;j<2650;j++);            //16MHz : 1msec
        }
}
void delay_us(unsigned int p){
        int i;
        for(i=0;i<p;i++);
}
void instruction_out(unsigned char b) {
        PORTA = b & 0xf0;;
        LCD_ENABLE;
        LCD_DISABLE;
```

```
	PORTA = (b<<4) & 0xf0;
	LCD_ENABLE;
	LCD_DISABLE;
	delay_us(700);
}
void char_out(unsigned char b) {
	PORTA = (b & 0xf0)|0x01;    //Upper 4 bit output
	LCD_ENABLE;
	LCD_DISABLE;
	PORTA = ((b<<4) & 0xf0)|0x01;          //Lower 4 bit output
	LCD_ENABLE;
	LCD_DISABLE;
	delay_us(100);
}
void string_out(unsigned char b, unsigned char *str)          // String output to LCD module
{
	unsigned int i=0;
	instruction_out(b);  //LCD location
	do{
		char_out(str[i]);
	}while(str[++i]!='\0');          //Until NULL
}
/*************************************************************
*	Initialize MOTOR
*
*************************************************************/
void Motor_Init(void){
	MOTOR_ENABLE;
	GoForward;
	while(checkButton2 != 1){
		motorDty = Rotary_Input;
		PWMDTY2 = motorDty;
	}
	checkButton2 = 0;
	MOTOR_DISABLE;
}
void SDcom_Initialize(void){
	unsigned char send[100];
	sprintf(send,"mode /m");       //Run SDcom on MCU mode
	delay_ms(50);
	sprintf(send,"fputs test.txt /a test:123");

	sendMsgSCI1(send);

}
void Read_Waypoint(void){
	unsigned char send[18];
	sprintf(send,"fread waypoint.txt");
	SDRead=1;
	sendMsgSCI1(send);
	wait_SDcom();
}
void wait_SDcom(void){
	while(sci1_temp=='\0');
	sci1_temp=0;
	delay_ms(50);
}

/*************************************************************
```

```
*           Process Waypoint
*           If everything done correctly, wayPoint.processOk is set to 1
*
*****************************************************************/
void Process_Waypoint(void){
        unsigned int i,j;
        //unsigned char string[10];
        unsigned char send[20];
        instruction_out(0x01);
        //string_out(0x80,wayPoint.Data);
    string_out(0x80,"Processing");
        do{
                //delay_ms(10);
                Parse_Waypoint(i);
                i++;
        }while(wayPoint.Data[i]!=0x23);                 // Process waypoint until meets '#'
        delay_ms(10);
        for(j=0;j<wayPoint.n_Points;j++){
                sprintf(send,"%d:%d%d,%d%d",j+1,    wayPoint.i_Lat[j][0],
        wayPoint.i_Lat[j][1],
        wayPoint.i_Lon[j][0],
        wayPoint.i_Lon[j][1]);
                switch(j){
                        case 0:
                                string_out(0xc0,send);
                                delay_ms(10);
                                break;
                        case 1:
                                string_out(0x94,send);delay_ms(10);
                                break;
                        case 2:
                                string_out(0xd4,send);delay_ms(10);
                                break;
                }
        }
        delay_ms(100);
        wayPoint.processOk=1;
}
/*****************************************************************
*           Find Way-point Information from data read from SD module
*           Repeat until '#'
*****************************************************************/
void Parse_Waypoint(int i){
        int j;
        unsigned char string[10];
#if SDCOM_TEST_MODE
        PORTB_BIT1=1;delay_ms(10);PORTB_BIT1=0;
#endif
        switch(wayPoint.state){
                case 0:
                        if(wayPoint.Data[i]==0x24)   {          //Found '$'
                                wayPoint.ptr=0;
                                wayPoint.state = 1;
                                wayPoint.n_Points++;
                        }
                        break;
                case 1:
                        wayPoint.Temporary[wayPoint.ptr++]=wayPoint.Data[i];
                        if(wayPoint.Data[i]==0x2E) { //Found '.'
```

```
                                        wayPoint.i_Lat[wayPoint.n_Points-1][0]  =  atoi(wayPoint.Temporary);
                                        wayPoint.state = 2;
                                        for(j=0;j<10;j++) wayPoint.Temporary[j]=' ';
                                        wayPoint.ptr=0;
                                        }
                                        break;
                                case 2:
                                        wayPoint.Temporary[wayPoint.ptr++]=wayPoint.Data[i];
                                        if(wayPoint.Data[i]==0x2C) {            //Found ','
                                                wayPoint.i_Lat[wayPoint.n_Points-1][1]                  =
atoi(wayPoint.Temporary);

                                                wayPoint.state = 3;
                                                wayPoint.ptr=0;
                                                for(j=0;j<10;j++) string[j]=' ';
                                        }
                                        break;
                                case 3:
                                        wayPoint.Temporary[wayPoint.ptr++]=wayPoint.Data[i];
                                        if(wayPoint.Data[i]==0x2E) {
                                                wayPoint.i_Lon[wayPoint.n_Points-1][0]                  =
atoi(wayPoint.Temporary);
                                                wayPoint.state = 4;
                                                wayPoint.ptr=0;
                                                for(j=0;j<10;j++) string[j]=' ';
                                                wayPoint.ptr=0;
                                        }
                                        break;
                                case 4:
                                        wayPoint.Temporary[wayPoint.ptr++]=wayPoint.Data[i];
                                        if(wayPoint.Data[i]=='*') {
                                                wayPoint.i_Lon[wayPoint.n_Points-1][1]                  =
atoi(wayPoint.Temporary);
                                                wayPoint.state = 0;
                                                wayPoint.ptr=0;
                                                for(j=0;j<10;j++) string[j]=' ';
                                        }
                                        break;
                                        }
}

void Parse_GPS(void){
        char        *strHead_RMC,*strTime_RMC,*strAOK_RMC,*strLat1_RMC,*strLat2_RMC,
*strNS_RMC, *strLon1_RMC,*strLon2_RMC, *strEW_RMC, *strSpd_RMC, *strHDG_RMC;
        unsigned char send[50];
        unsigned int i_Lat1, i_Lat2, i_Lon1, i_Lon2, i_Distance;
        int D_LAT, D_LON,Goal_HDG, D_HDG, Current_HDG, Current_SPD, TURN_ANGLE;
        float D_Angle;
        strHead_RMC     = strtok(gpsData.Temp,",");
        strTime_RMC     = strtok(0,",");
        strAOK_RMC          = strtok(0,",");
        strLat1_RMC      = strtok(0,".");
        strLat2_RMC      = strtok(0,",");
        strNS_RMC        = strtok(0,",");
        strLon1_RMC      = strtok(0,".");
        strLon2_RMC      = strtok(0,",");
        strEW_RMC        = strtok(0,",");
        strSpd_RMC       = strtok(0,",");
        strHDG_RMC = strtok(0,",");
```

```
		gpsData.NS = *strNS_RMC;
		gpsData.EW = *strEW_RMC;
		i_Lat1= atoi(strLat1_RMC);
		i_Lat2= atoi(strLat2_RMC);
		i_Lon1= atoi(strLon1_RMC);
		i_Lon2= atoi(strLon2_RMC);
		if(i_Lat1<1) {
			string_out(line_1,"!! GPS invalid !!");
			return;
		}
		Current_SPD = atoi(strSpd_RMC);
		Current_HDG = atoi(strHDG_RMC);
		sprintf(send,"%c%4d.%4d%c%4d.%4d",*strNS_RMC,i_Lat1,i_Lat2,*strEW_RMC,i_Lon1,i_Lon2);
		string_out(line_1,send);
//#if GPS_RECORD_MODE
if(onGPS_record){
		sprintf(send,"fputs test.txt /a $%d.%d,%d.%d",i_Lat1,i_Lat2,i_Lon1,i_Lon2);
		sendMsgSCI1(send);
}
//#endif
		sprintf(send,"HDG:%3d SPD:%3d",Current_HDG,Current_SPD);
		//string_out(line_2,send);
		gpsData.rcvCmpl = 1;
		if(gpsData.d_Lat[0]<0){
			string_out(line_1,"!! GPS invalid !!");
			return;
		}
		else {
			D_LAT = (wayPoint.i_Lat[Navi.Route][0] - i_Lat1) * 1000 + (int)
(wayPoint.i_Lat[Navi.Route][1] - i_Lat2)/10;
			D_LON = (i_Lon1 - wayPoint.i_Lon[Navi.Route][0] )*1000+(int)(i_Lon2 -
wayPoint.i_Lon[Navi.Route][1])/10;
			i_Distance = sqrtf(D_LAT*D_LAT + D_LON*D_LON);	//Calculate Distance
			Goal_HDG = (atan2f(D_LON,D_LAT)*180/PI);
			if(D_LON<0) Goal_HDG +=360;
			if(Goal_HDG - Current_HDG > 180) D_HDG = Goal_HDG - 360 - Current_HDG;
			else if(Goal_HDG - Current_HDG < -180) D_HDG = 360-Current_HDG + Goal_HDG;
			else D_HDG = Goal_HDG - Current_HDG;

			sprintf(send,"#%2d:Y%5d X%5d\0",Navi.Route,D_LAT,D_LON);
			string_out(line_2,send);
			sprintf(send,"gHD%4ddHD%4d %4d\0",Goal_HDG, D_HDG,i_Distance);
			string_out(line_3,send);

			if(D_HDG>5 | D_HDG<-5) {
				TURN_ANGLE = 1550 - D_HDG*2;			//Turn to RIGHT
				Goal_RPM = 150;
			}else {
				TURN_ANGLE = 1550;
				Goal_RPM = 200;
			}
		if(onAvoidance){		//Obstacle Avoidance
				TURN_ANGLE += Obstacle.Maneuver;
				Goal_RPM += Obstacle.Decellerate;
		}
			if(TURN_ANGLE>1950) TURN_ANGLE = 1950;			//Servo Limit
			else if(TURN_ANGLE<1230) TURN_ANGLE = 1330;
			if(Goal_RPM < 1) Goal_RPM = 1;			//Motor Limit
```

```
                        ANGLE_TIME =TURN_ANGLE;

                        if(i_Distance < 2) {            // if the vehicle almost reaches the waypoint
                                if(Navi.Route > wayPoint.n_Points){
                                        //IF the Vehicle reaches the final point then STOP
                                        Goal_RPM = 0;
                                        MOTOR_DISABLE;
                                }else Navi.Route++;
                        }
                }
}
void PID_Control(int Current_RPM){
                unsigned char send[20];
                int Current_RPM_Error, Current_Time = 0;
                int PID_Result;
#if PID_TEST_MODE
                Goal_RPM = Rotary_Input/4;                      // For Test
                P_Gain = P_Tuner / 100;
#endif
                Current_RPM_Error = Goal_RPM - Current_RPM;
#if ON_OFF_CONTROL_MODE                 // Look like the most stable controller
                if(Current_RPM_Error > 5 && Current_RPM_Error<50) motorDty++;
                else if(Current_RPM_Error > 25)    motorDty+=3;
                else if(Current_RPM_Error < -5 && Current_RPM_Error>-50) motorDty--;
                else if(Current_RPM_Error <-25) motorDty-=3;
                if(motorDty>255) motorDty = 255;
                else if(motorDty<0) motorDty = 0;
                PWMDTY2 = motorDty;
#else if PID_CONTROL_MODE
                PID_Result = P_Gain * Current_RPM_Error;
                if(PID_Result>250) PID_Result = 250;
                else if (PID_Result<0) PID_Result = 0;
                PWMDTY2 = PID_Result;
#endif
#if PID_TEST_MODE
                        sprintf(send,"SON%3d L%4d R%4d",PWMDTY7,SONAR_1, SONAR_2);
                        if(Encoder_Display_Enable) string_out(line_1, send);
                        sprintf(send,"Kp:%4d,Svo:%4d",P_Tuner,ANGLE_TIME);
                string_out(line_2,send);
                sprintf(send,"Go:%4d|PID:%4d",Goal_RPM,motorDty);
                string_out(line_3,send);
#endif
#if PID_RECORD_MODE             //write input and output data in file
                sprintf(send,"fputs test.txt /a %d,%d;",Goal_RPM,Current_RPM);
                sendMsgSCI1(send);
#endif
                //Old_RPM_Error2 = Old_RPM_Error1;
                //Old_RPM_Error1 = Current_RPM_Error;
                //Old_Time = Current_Time;
                //return;
}
void main(void) {
  /* put your own code here */
                unsigned char send[20];
                int i,j,i_Dim;
                unsigned int Sonar_Count,Sonar_Temp_L,Sonar_Temp_R,Correct;
                RTIInit();
                Timer_Init();
                PORT_Init();
```

```
ATD_Init();
PWM_Init();
gpsData.state=0;
SCI1_Init();
startTimeBase();
LCD_Init();
instruction_out(0x01);
string_out(0x80, "LCD HELLO WORLD");
instruction_out(0x02);
string_out(0xC0, "Kyubot");
SDcom_Initialize();
instruction_out(0x01);          //LCD CLEAR
string_out(0x80, "SDcom Init!");
delay_ms(100);

Read_Waypoint();                // Send Instruction to SD-COMM
while(wayPoint.readOk!=1){
        }     //Wait until waypoint data read finished

instruction_out(0x01);
string_out(0x80,"Read Success!!");
delay_ms(500);
while(!wayPoint.processOk){
        Process_Waypoint();                     // Process Waypoint
}
while(checkButton2!=1){
        string_out(line_1,"Press the Button");
        delay_ms(200);
        string_out(line_1,"                    ");
        delay_ms(200);
}
checkButton2 = 0;
instruction_out(LCD_CLEAR);
delay_ms(100);

string_out(line_1," ===== SET UP =====");
        if(PTH_PTH7 == 1) { //if DIP switch is set to OFF
                string_out(line_2,"GPS record mode OFF");
                onGPS_record = 0;
        }else {
                string_out(line_2,"GPS record mode ON ");
                onGPS_record = 1;
        }
        if(PTH_PTH6 == 1) {
                string_out(line_3,"PID control mode OFF");
                onPID_Control = 0;
        }else {
                string_out(line_3,"PID control mode ON ");
                onPID_Control = 1;
        }
        if(PTH_PTH5 == 1) {
                string_out(line_4,"Obstacle: Ignore");
                onAvoidance = 0;
        }else {
                string_out(line_4,"Obstacle: Avoid ");
                onAvoidance = 1;
        }
i_Dim=2;
while(checkButton2!=1){
```

```
                    if(PTH_PTH7 == 1 && onGPS_record == 1) { //if DIP switch is set to OFF
                            string_out(line_2,"GPS record mode OFF");
                            onGPS_record = 0;
                    }
                    else if(PTH_PTH7== 0 && onGPS_record == 0){
                            PORTB_BIT0 = 0;//string_out(line_1,"DIP7 is ON");
                            string_out(line_2,"GPS record mode ON ");
                            onGPS_record = 1;
                    }
                    if(PTH_PTH6 == 1 && onPID_Control == 1) {
                            string_out(line_3,"PID control mode OFF");
                            onPID_Control = 0;
                    }else if(PTH_PTH6 ==0 && onPID_Control == 0){
                            string_out(line_3,"PID control mode ON ");
                            onPID_Control = 1;
                    }
                    if(PTH_PTH5 == 1 && onAvoidance ==1) {
                            string_out(line_4,"Obstacle: Ignore");
                            onAvoidance = 0;
                    }else if(PTH_PTH5 == 0 && onAvoidance == 0) {
                            string_out(line_4,"Obstacle: Avoid ");
                            onAvoidance = 1;
                    }
                    if(PWMDTY6==254){
                            i_Dim=-1;
                    }else if(PWMDTY6==0){
                            i_Dim=1;
                            delay_ms(40);
                    }
                    //i_Light +=i_Dim
                    PWMDTY6 +=i_Dim;
                    delay_ms(1);

            }                               //Wait until button2 is pushed
            checkButton2=0;
            delay_ms(100);
            SDRead=0;
#endif
            //Timer_Init();
            MOTOR_ENABLE;
            GoForward;
            PID_Enable = 1;

            instruction_out(0x01);
            instruction_out(0x01);
            string_out(line_1,"GPS is ON!");
            SCI0_Init();                    // START receiving GPS data!
            PWMDTY6 = 0xFF;
            //PTP_PTP6 = 1;
            //Encoder_Enable=1;
            //i=0;      //for test
            Sonar_Count = 0;
            j=0;
            PWMDTY7 = Sonar_Center;
            for(;;){
            if(gpsData.GPS_Received == 1){
                    Parse_GPS();                    // IF GPGGA code has been parsed out
                    gpsData.GPS_Received = 0;
                    }
```

```
if(onAvoidance){

                    if(Sonar_Count < 10){
                              Sonar_Temp_L += SONAR_1;
                              Sonar_Temp_R += SONAR_2;
                              Sonar_Count++;
                    }else {
                              Sonar_L =Sonar_Temp_L/10;
                              Sonar_R =Sonar_Temp_R/10;
                              Sonar_Count = 0;
                              Sonar_Temp_L=0;
                              Sonar_Temp_R=0;



                              if(Sonar_L<40 ){
                                        //Obstacle.Distance[j] = Sonar_L;
                                        PTJ_PTJ1 = 1;
                                        //GoForward;
                                        //Goal_RPM = 80;
                                        //CURRENT_ANGLE = ANGLE_TIME;
                                        //ANGLE_TIME =1350;
                                        Obstacle.Decellerate = -1000/Sonar_L;
                                        Obstacle.Maneuver = -2500/Sonar_L;
                                        Correct = 2500/Sonar_L;
                              }else{
                                        PTJ_PTJ0 = 0;
                                        PTJ_PTJ1 = 0;
                                        GoForward;
                                        Correct = 0;
                              }
                              sprintf(send,"SNR:%3d CR:%3d \0",Sonar_L, Correct);
                              string_out(line_4,send);
                    }
}}
}
```