```c
//#############################################################################
// FILE:   LAB7starter_main.c
//
// TITLE:  Lab Starter
//#############################################################################

// Included Files
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <string.h>
#include <math.h>
#include <limits.h>
#include "F28x_Project.h"
#include "driverlib.h"
#include "device.h"
#include "F28379dSerial.h"
#include "LEDPatterns.h"
#include "song.h"
#include "dsp.h"
#include "fpu32/fpu_rfft.h"

#define PI          3.1415926535897932384626433832795
#define TWOPI       6.283185307179586476925286766559
#define HALFPI      1.5707963267948966192313216916398
// The Launchpad's CPU Frequency set to 200 you should not change this value
#define LAUNCHPAD_CPU_FREQUENCY 200

// ----- code for CAN start here -----
#include "F28379dCAN.h"
//#define TX_MSG_DATA_LENGTH    4
//#define TX_MSG_OBJ_ID         0  //transmit

#define RX_MSG_DATA_LENGTH    8
#define RX_MSG_OBJ_ID_1       1  //measurement from sensor 1
#define RX_MSG_OBJ_ID_2       2  //measurement from sensor 2
#define RX_MSG_OBJ_ID_3       3  //quality from sensor 1
#define RX_MSG_OBJ_ID_4       4  //quality from sensor 2
// ----- code for CAN end here -----

// Interrupt Service Routines predefinition
__interrupt void cpu_timer0_isr(void);
__interrupt void cpu_timer1_isr(void);
__interrupt void cpu_timer2_isr(void);
__interrupt void SWI_isr(void);
// ----- code for CAN start here -----
__interrupt void can_isr(void);
// ----- code for CAN end here -----
__interrupt void SPIB_isr(void);
void setupSpib(void);

// Count variables
uint32_t numTimer0calls = 0;
uint32_t numSWIcalls = 0;
extern uint32_t numRXA;
```

```
uint16_t UARTPrint = 0;
uint16_t LEDdisplaynum = 0;
int16_t accelx_raw = 0;
int16_t accely_raw = 0;
int16_t accelz_raw = 0;
int16_t gyrox_raw = 0;
int16_t gyroy_raw = 0;
int16_t gyroz_raw = 0;

float accelx = 0;
float accely = 0;
float accelz = 0;

float gyrox = 0;
float gyroy = 0;
float gyroz = 0;

float LeftWheel = 0.0;
float RightWheel = 0.0;
float LeftWheelmeter =0.0;
float RightWheelmeter =0.0;
float uleft = 5.0;
float uright = 5.0;
float PosLeft_k =0.0;
float PosLeft_k_1 =0.0;
float PosRt_k =0.0;
float PosRt_k_1 =0.0;
float VLeftK =0.0;
float VRtK =0.0;
float eleftk =0.0;
float vref =0.25;
float eleftk_1 = 0.0;
float Ileftk =0.0;
float Ileftk_1=0.0;
float ertk =0.0;
float ertk_1 = 0.0;
float Irtk =0.0;
float Irtk_1=0.0;
float Ki =20.0;
float Kp=3.0;
float Kd =0.08;
float turn =0.0;
float eturn =0.0;
float kpturn =3.0;
float radius = 0.06;
float widthrob =0.18;
float beavg = 0.0;
float beavgdot =0.0;
float xrdot =0.0;
float yrdot = 0.0;
float angvel_left =0.0;
float angvel_rt=0.0;
float x_1 = 0.0;
float x_1dot= 0.0;
float y_1 =0.0;
```

```c
float y_1dot=0.0;

float distright =0.0;
float distfront =0.0;
float distright_1 = 0.0;
float distfront_1 =0.0;
float kprt = 0.0015;
float kpft =0.0005;
float refrt =300;
float refft =1400;
float threshold1=300;
float threshold2=500;
float rtwallfollow =0.0;


float xka2=0.0;
float xka3=0.0;
float ykb1=0.0;

int32_t ADCA1count =0;
int16_t adca2result =0;
int16_t adca3result =0;
float ltangle =0.0;
float rtangle =0.0;

float printLV3 = 0;
float printLV4 = 0;
float printLV5 = 0;
float printLV6 = 0;
float printLV7 = 0;
float printLV8 = 0;
float x = 0;
float y = 0;
float bearing = 0;

// Needed global Variables
float accelx_offset = 0;
float accely_offset = 0;
float accelz_offset = 0;
float gyrox_offset = 0;
float gyroy_offset = 0;
float gyroz_offset = 0;
float accelzBalancePoint = -0.685;
int16 IMU_data[9];
uint16_t temp=0;
int16_t doneCal = 0;
float tilt_value = 0;
float tilt_array[4] = {0, 0, 0, 0};
float gyro_value = 0;
float gyro_array[4] = {0, 0, 0, 0};
float LeftWheelArray[4] = {0,0,0,0};
float RightWheelArray[4] = {0,0,0,0};
// Kalman Filter vars
float T = 0.001; //sample rate, 1ms
float Q = 0.01; // made global to enable changing in runtime
```

```
float R = 25000;//50000;
float kalman_tilt = 0;
float kalman_P = 22.365;
int16_t SpibNumCalls = -1;
float pred_P = 0;
float kalman_K = 0;
int32_t timecount = 0;
int16_t calibration_state = 0;
int32_t calibration_count = 0;
float vel_Right =0.0;
float vel_Left=0.0;
float vel_Right_pre =0.0;
float vel_Left_pre =0.0;
float ltwheel_pre=0.0;
float rtwheel_pre=0.0;
float gyroratedot_pre=0.0;
float gyrovalue_pre =0.0;
float gyrorate_dot=0.0;
float k1=-60;
float k2=-4.5;
float k3=-1.1;
float k4=-0.1;
float ubal =0.0;
float whldiff =0.0;
float whldiff_1=0.0;
float vel_whldiff =0.0;
float vel_whldiff_1=0.0;
float turnref=0.0;
float errordiff=0.0;
float errordiff_1 =0.0;
float intdiff =0.0;
float intdiff_1=0.0;
float turnrate =0.0;
float turnrate_1=0.0;
float turnref_1=0.0;
float avgwheelvel=0.0;
float espeed=0.0;
float Segbot_refspeed=0.0;
float espeed_1=0.0;
float IK_espeed=0.0;
float IK_espeed_1=0.0;
float forwardbackwardcommand =0.0;
float kpspeed =0.35;
float kispeed =1.5;



extern uint16_t NewLVData;
extern float fromLVvalues[LVNUM_TOFROM_FLOATS];
extern LVSendFloats_t DataToLabView;
extern char LVsenddata[LVNUM_TOFROM_FLOATS*4+2];
extern uint16_t newLinuxCommands;
extern float LinuxCommands[CMDNUM_FROM_FLOATS];
```

```
//int32_t SpibNumCalls = 0;

// ----- code for CAN start here -----
// volatile uint32_t txMsgCount = 0;
// extern uint16_t txMsgData[4];

volatile uint32_t rxMsgCount_1 = 0;
volatile uint32_t rxMsgCount_3 = 0;
extern uint16_t rxMsgData[8];

uint32_t dis_raw_1[2];
uint32_t dis_raw_3[2];
uint32_t dis_1 = 0;
uint32_t dis_3 = 0;

uint32_t quality_raw_1[4];
uint32_t quality_raw_3[4];
float quality_1 = 0.0;
float quality_3 = 0.0;

uint32_t lightlevel_raw_1[4];
uint32_t lightlevel_raw_3[4];
float lightlevel_1 = 0.0;
float lightlevel_3 = 0.0;


uint32_t measure_status_1 = 0;
uint32_t measure_status_3 = 0;

volatile uint32_t errorFlag = 0;
// ----- code for CAN end here -----
//JS copy from the guideline
void init_eQEPs(void) {
    // setup eQEP1 pins for input
    EALLOW;
    //Disable internal pull-up for the selected output pins for reduced power
consumption
    GpioCtrlRegs.GPAPUD.bit.GPIO20 = 1; // Disable pull-up on GPIO20 (EQEP1A)
    GpioCtrlRegs.GPAPUD.bit.GPIO21 = 1; // Disable pull-up on GPIO21 (EQEP1B)
    GpioCtrlRegs.GPAQSEL2.bit.GPIO20 = 2; // Qual every 6 samples
    GpioCtrlRegs.GPAQSEL2.bit.GPIO21 = 2; // Qual every 6 samples
    EDIS;
    // This specifies which of the possible GPIO pins will be EQEP1 functional pins.
    // Comment out other unwanted lines.
    GPIO_SetupPinMux(20, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinMux(21, GPIO_MUX_CPU1, 1);
    EQep1Regs.QEPCTL.bit.QPEN = 0; // make sure eqep in reset
    EQep1Regs.QDECCTL.bit.QSRC = 0; // Quadrature count mode
    EQep1Regs.QPOSCTL.all = 0x0; // Disable eQep Position Compare
    EQep1Regs.QCAPCTL.all = 0x0; // Disable eQep Capture
    EQep1Regs.QEINT.all = 0x0; // Disable all eQep interrupts
    EQep1Regs.QPOSMAX = 0xFFFFFFFF; // use full range of the 32 bit count
    EQep1Regs.QEPCTL.bit.FREE_SOFT = 2; // EQep uneffected by emulation suspend in
Code Composer
    EQep1Regs.QPOSCNT = 0;
```

```
    EQep1Regs.QEPCTL.bit.QPEN = 1; // Enable EQep
    // setup QEP2 pins for input
    EALLOW;
    //Disable internal pull-up for the selected output pinsfor reduced power
consumption
    GpioCtrlRegs.GPBPUD.bit.GPIO54 = 1; // Disable pull-up on GPIO54 (EQEP2A)
    GpioCtrlRegs.GPBPUD.bit.GPIO55 = 1; // Disable pull-up on GPIO55 (EQEP2B)
    GpioCtrlRegs.GPBQSEL2.bit.GPIO54 = 2; // Qual every 6 samples
    GpioCtrlRegs.GPBQSEL2.bit.GPIO55 = 2; // Qual every 6 samples
    EDIS;
    GPIO_SetupPinMux(54, GPIO_MUX_CPU1, 5); // set GPIO54 and eQep2A
    GPIO_SetupPinMux(55, GPIO_MUX_CPU1, 5); // set GPIO54 and eQep2B
    EQep2Regs.QEPCTL.bit.QPEN = 0; // make sure qep reset
    EQep2Regs.QDECCTL.bit.QSRC = 0; // Quadrature count mode
    EQep2Regs.QPOSCTL.all = 0x0; // Disable eQep Position Compare
    EQep2Regs.QCAPCTL.all = 0x0; // Disable eQep Capture
    EQep2Regs.QEINT.all = 0x0; // Disable all eQep interrupts
    EQep2Regs.QPOSMAX = 0xFFFFFFFF; // use full range of the 32 bit count.
    EQep2Regs.QEPCTL.bit.FREE_SOFT = 2; // EQep uneffected by emulation suspend
    EQep2Regs.QPOSCNT = 0;
    EQep2Regs.QEPCTL.bit.QPEN = 1; // Enable EQep
}

//JS define array use to calculate filter signals
float xkb1array[101]={};
//JS copy from guideline
float readEncLeft(void) {
    int32_t raw = 0;
    uint32_t QEP_maxvalue = 0xFFFFFFFFU; //4294967295U
    raw = EQep1Regs.QPOSCNT;
    if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue; // I don't think this is needed
and never true
    // 100 slits in the encoder disk so 100 square waves per one revolution of the
    // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so
400 counts per one rev
    // of the DC motor's back shaft. Then the gear motor's gear ratio is 30:1.
    //JS the number of radians the wheel turn 400 counts per wheel revolution * 30
motor rotations
    return (raw*(2*PI/(30*400)));
}
float readEncRight(void) {
    int32_t raw = 0;
    uint32_t QEP_maxvalue = 0xFFFFFFFFU; //4294967295U -1 32bit signed int
    raw = EQep2Regs.QPOSCNT;
    if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue; // I don't think this is needed
and never true
    // 100 slits in the encoder disk so 100 square waves per one revolution of the
    // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so
400 counts per one rev
    // of the DC motor's back shaft. Then the gear motor's gear ratio is 30:1.
    return (raw*(2*PI/(30*400)));
}
//JS lab 7 exercise 1.2, copy function setEPWM2A and setEPWM2B
//JS saturate controleffort between -10 to 10 for right motor
void setEPWM2A(float controleffort)
```

```
{
    if (controleffort > 10){
        controleffort = 10;
    }
    if (controleffort < -10){
        controleffort = -10;
    }
    //JS control duty cycle to 0% when controleffort is -10, 50% when controleffort
is 0, 100% when controleffort is 10, converts controleffort value between -10 and 10
to duty cycle 0% to 100%
    EPwm2Regs.CMPA.bit.CMPA = ((controleffort + (float)10))/ ((float)20)*
EPwm2Regs.TBPRD;
}
//JS saturate controleffort between -10 to 10 for left motor
void setEPWM2B(float controleffort){
    if (controleffort > 10){
        controleffort = 10;
    }
    if (controleffort < -10){
        controleffort = -10;
    }
    //JS control duty cycle to 0% when controleffort is -10, 50% when controleffort
is 0, 100% when controleffort is 10, converts controleffort value between -10 and 10
to duty cycle 0% to 100%
    EPwm2Regs.CMPB.bit.CMPB = ((controleffort + (float)10))/ ((float)20)*
EPwm2Regs.TBPRD;
}
__interrupt void ADCA_ISR (void) {
    //JS lab7 exercise 1.1, SPI transmission code copy from lab 6
    adca2result = AdcaResultRegs.ADCRESULT0;
    adca3result = AdcaResultRegs.ADCRESULT1;

    //JS lab 7 exercise 1.4, convert to volts and store to global variables
    xka2 = (adca2result/4096.0)*3.0;
    xka3 = (adca3result/4096.0)*3.0;
    // Print  voltage value to TeraTerm every 100ms
    ADCA1count++;
    if (ADCA1count % 100 == 0){
        UARTPrint = 1;
    }
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
    //JS lab7 exercise1.5, SPI transmission and reception of readings
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPIFFRX.bit.RXFFIL = 8;
    SpibRegs.SPITXBUF = 0xBA00;
    SpibRegs.SPITXBUF = 0x0000;
    SpibRegs.SPITXBUF = 0x0000;
    SpibRegs.SPITXBUF = 0x0000;
    SpibRegs.SPITXBUF = 0x0000;
    SpibRegs.SPITXBUF = 0x0000;
    SpibRegs.SPITXBUF = 0x0000;
    SpibRegs.SPITXBUF = 0x0000;
}
```

```c
void main(void)
{
    // PLL, WatchDog, enable Peripheral Clocks
    // This example function is found in the F2837xD_SysCtrl.c file.
    InitSysCtrl();

    InitGpio();

    //JS PinMux to set GPIO2 to EPWM2A
    GPIO_SetupPinMux(2, GPIO_MUX_CPU1, 1);
    //JS PinMux to set GPIO3 to EPWM2B
    GPIO_SetupPinMux(3, GPIO_MUX_CPU1, 1);

    // Blue LED on LaunchPad
    GPIO_SetupPinMux(31, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(31, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPASET.bit.GPIO31 = 1;

    // Red LED on LaunchPad
    GPIO_SetupPinMux(34, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(34, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPBSET.bit.GPIO34 = 1;

    // LED1 and PWM Pin
    GPIO_SetupPinMux(22, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(22, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;

    // LED2
    GPIO_SetupPinMux(94, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(94, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPCCLEAR.bit.GPIO94 = 1;

    // LED3
    GPIO_SetupPinMux(95, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(95, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPCCLEAR.bit.GPIO95 = 1;

    // LED4
    GPIO_SetupPinMux(97, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(97, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPDCLEAR.bit.GPIO97 = 1;

    // LED5
    GPIO_SetupPinMux(111, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(111, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPDCLEAR.bit.GPIO111 = 1;

    // LED6
    GPIO_SetupPinMux(130, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(130, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPECLEAR.bit.GPIO130 = 1;

    // LED7
    GPIO_SetupPinMux(131, GPIO_MUX_CPU1, 0);
```

```
GPIO_SetupPinOptions(131, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO131 = 1;

// LED8
GPIO_SetupPinMux(25, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(25, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;

// LED9
GPIO_SetupPinMux(26, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(26, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO26 = 1;

// LED10
GPIO_SetupPinMux(27, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(27, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;

// LED11
GPIO_SetupPinMux(60, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(60, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1;

// LED12
GPIO_SetupPinMux(61, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(61, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;

// LED13
GPIO_SetupPinMux(157, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(157, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO157 = 1;

// LED14
GPIO_SetupPinMux(158, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(158, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO158 = 1;

// LED15
GPIO_SetupPinMux(159, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(159, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPECLEAR.bit.GPIO159 = 1;

// LED16
GPIO_SetupPinMux(160, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(160, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPFCLEAR.bit.GPIO160 = 1;

//WIZNET Reset
GPIO_SetupPinMux(0, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(0, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO0 = 1;

//ESP8266 Reset
GPIO_SetupPinMux(1, GPIO_MUX_CPU1, 0);
```

```
GPIO_SetupPinOptions(1, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO1 = 1;

//SPIRAM  CS  Chip Select
GPIO_SetupPinMux(19, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(19, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO19 = 1;

//DRV8874 #1 DIR  Direction
GPIO_SetupPinMux(29, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(29, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO29 = 1;

//DRV8874 #2 DIR  Direction
GPIO_SetupPinMux(32, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(32, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPBSET.bit.GPIO32 = 1;

//DAN28027  CS  Chip Select
GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPASET.bit.GPIO9 = 1;

//MPU9250  CS  Chip Select
GPIO_SetupPinMux(66, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(66, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPCSET.bit.GPIO66 = 1;

//WIZNET  CS  Chip Select
GPIO_SetupPinMux(125, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(125, GPIO_OUTPUT, GPIO_PUSHPULL);
GpioDataRegs.GPDSET.bit.GPIO125 = 1;

//PushButton 1
GPIO_SetupPinMux(4, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(4, GPIO_INPUT, GPIO_PULLUP);

//PushButton 2
GPIO_SetupPinMux(5, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(5, GPIO_INPUT, GPIO_PULLUP);

//PushButton 3
GPIO_SetupPinMux(6, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(6, GPIO_INPUT, GPIO_PULLUP);

//PushButton 4
GPIO_SetupPinMux(7, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(7, GPIO_INPUT, GPIO_PULLUP);

//Joy Stick Pushbutton
GPIO_SetupPinMux(8, GPIO_MUX_CPU1, 0);
GPIO_SetupPinOptions(8, GPIO_INPUT, GPIO_PULLUP);

// ----- code for CAN start here -----
//GPIO17 - CANRXB
```

```
    GPIO_SetupPinMux(17, GPIO_MUX_CPU1, 2);
    GPIO_SetupPinOptions(17, GPIO_INPUT, GPIO_ASYNC);

    //GPIO12 - CANTXB
    GPIO_SetupPinMux(12, GPIO_MUX_CPU1, 2);
    GPIO_SetupPinOptions(12, GPIO_OUTPUT, GPIO_PUSHPULL);
    // ----- code for CAN end here -----



    // ----- code for CAN start here -----
    // Initialize the CAN controller
    InitCANB();

    // Set up the CAN bus bit rate to 1000 kbps
    setCANBitRate(200000000, 1000000);

    // Enables Interrupt line 0, Error & Status Change interrupts in CAN_CTL
register.
    CanbRegs.CAN_CTL.bit.IE0= 1;
    CanbRegs.CAN_CTL.bit.EIE= 1;
    // ----- code for CAN end here -----

    // Clear all interrupts and initialize PIE vector table:
    // Disable CPU interrupts
    DINT;

    // Initialize the PIE control registers to their default state.
    // The default state is all PIE interrupts disabled and flags
    // are cleared.
    // This function is found in the F2837xD_PieCtrl.c file.
    InitPieCtrl();

    // Disable CPU interrupts and clear all CPU interrupt flags:
    IER = 0x0000;
    IFR = 0x0000;

    // Initialize the PIE vector table with pointers to the shell Interrupt
    // Service Routines (ISR).
    // This will populate the entire table, even if the interrupt
    // is not used in this example.  This is useful for debug purposes.
    // The shell ISR routines are found in F2837xD_DefaultIsr.c.
    // This function is found in F2837xD_PieVect.c.
    InitPieVectTable();

    // Interrupts that are used in this example are re-mapped to
    // ISR functions found within this project
    EALLOW;  // This is needed to write to EALLOW protected registers
    PieVectTable.TIMER0_INT = &cpu_timer0_isr;
    PieVectTable.TIMER1_INT = &cpu_timer1_isr;
    PieVectTable.TIMER2_INT = &cpu_timer2_isr;
    PieVectTable.SCIA_RX_INT = &RXAINT_recv_ready;
    PieVectTable.SCIB_RX_INT = &RXBINT_recv_ready;
    PieVectTable.SCIC_RX_INT = &RXCINT_recv_ready;
    PieVectTable.SCID_RX_INT = &RXDINT_recv_ready;
```

```
    PieVectTable.SCIA_TX_INT = &TXAINT_data_sent;
    PieVectTable.SCIB_TX_INT = &TXBINT_data_sent;
    PieVectTable.SCIC_TX_INT = &TXCINT_data_sent;
    PieVectTable.SCID_TX_INT = &TXDINT_data_sent;
    PieVectTable.SPIB_RX_INT = &SPIB_isr;
    //JS for ADCA to call the memory address of ADCA_ISR
    PieVectTable.ADCA1_INT = &ADCA_ISR;

    PieVectTable.EMIF_ERROR_INT = &SWI_isr;
    // ----- code for CAN start here -----
    PieVectTable.CANB0_INT = &can_isr;
    // ----- code for CAN end here -----
    EDIS;    // This is needed to disable write to EALLOW protected registers


    // Initialize the CpuTimers Device Peripheral. This function can be
    // found in F2837xD_CpuTimers.c
    InitCpuTimers();

    // Configure CPU-Timer 0, 1, and 2 to interrupt every given period:
    // 200MHz CPU Freq,                       Period (in uSeconds)
    ConfigCpuTimer(&CpuTimer0, LAUNCHPAD_CPU_FREQUENCY, 4000);
    ConfigCpuTimer(&CpuTimer1, LAUNCHPAD_CPU_FREQUENCY, 20000);
    ConfigCpuTimer(&CpuTimer2, LAUNCHPAD_CPU_FREQUENCY, 40000);

    // Enable CpuTimer Interrupt bit TIE
    CpuTimer0Regs.TCR.all = 0x4000;
    CpuTimer1Regs.TCR.all = 0x4000;
    CpuTimer2Regs.TCR.all = 0x4000;

    init_serialSCIA(&SerialA,115200);
    //JS lab7 exercise 1.3, initialize SPIB
    setupSpib();

    //JS Count up Mode bit is 00
    EPwm2Regs.TBCTL.bit.CTRMODE = 0;
    //JS 2/3 free run bit is 1x, 10 or 11
    EPwm2Regs.TBCTL.bit.FREE_SOFT = 2;
    //JS disable the phase loading bit is 0
    EPwm2Regs.TBCTL.bit.PHSEN = 0;
    //JS CLKDIV is 1, 2 to the power of 0
    EPwm2Regs.TBCTL.bit.CLKDIV = 0;
    //JS Start the timer at 0
    EPwm2Regs.TBCTR = 0;
    //JS Signal needs to be 20KHz, to have a period of 50 microseconds, TBPRD value
get divided by carrier frequency
    EPwm2Regs.TBPRD = 2500;
    //JS 0%*TBPRD for duty cycle
    EPwm2Regs.CMPA.bit.CMPA = 0;
    //JS needs CMPB for EPWM2B
    EPwm2Regs.CMPB.bit.CMPB = 0;
    EPwm2Regs.AQCTLA.bit.CAU = 1;
    //JS needs CBU for EPWM2B
    EPwm2Regs.AQCTLB.bit.CBU = 1;
    EPwm2Regs.AQCTLA.bit.ZRO = 2;
```

```
    //JS needs AQCTLB.ZRO for EPWM2B
    EPwm2Regs.AQCTLB.bit.ZRO = 2;
    EPwm2Regs.TBPHS.bit.TBPHS = 0;

    //JS EPWM5 and ADCB for setup microphone
    EALLOW;
    EPwm5Regs.ETSEL.bit.SOCAEN = 0; // Disable SOC on A group
    EPwm5Regs.TBCTL.bit.CTRMODE = 3; // freeze counter
    //JS only bit 1 is high,Enable event time-base counter equal to period
    EPwm5Regs.ETSEL.bit.SOCASEL = 2; // Select Event when counter equal to PRD
    //JS when only bit 0 is hig, Generate the EPWM5SOCA pulse on the first event
    EPwm5Regs.ETPS.bit.SOCAPRD = 1; // Generate pulse on 1st event ("pulse" is the
same as "trigger")
    EPwm5Regs.TBCTR = 0x0; // Clear counter
    EPwm5Regs.TBPHS.bit.TBPHS = 0x0000; // Phase is 0
    EPwm5Regs.TBCTL.bit.PHSEN = 0; // Disable phase loading
    EPwm5Regs.TBCTL.bit.CLKDIV = 0; // divide by 1 50Mhz Clock
    //JS lab 7 exercise 1.1, trigger every 1ms
    EPwm5Regs.TBPRD = 50000;
    // Notice here that we are not setting CMPA or CMPB because we are not using the
PWM signal
    EPwm5Regs.ETSEL.bit.SOCAEN = 1; //enable SOCA
    //JS up-count mode, set it to 0
    EPwm5Regs.TBCTL.bit.CTRMODE = 0; //unfreeze, and enter up count mode
    EDIS;


    EALLOW;
    //write configurations for all ADCs ADCA, ADCB, ADCC, ADCD

    AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read
calibration settings

    //Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    //power up the ADCs
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    DELAY_US(1000);



    //ADCA
    //JS set SOC0 to pin2
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2; //SOC0 will convert Channel you choose Does
not have to be A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles =
500ns
    //JS EPWM5 ADCSOCA is 13
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 13;// EPWM5 ADCSOCA or another trigger you
choose will trigger SOC0
    //JS set SOC1 to pin3
    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3; //SOC1 will convert Channel you choose Does
not have to be A1
```

```
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = 99; //sample window is acqps + 1 SYSCLK cycles = 
500ns
    //JS EPWM5 ADCSOCA is 13
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 13;// EPWM5 ADCSOCA or another trigger you 
choose will trigger SOC1
    //JS set to the last converted SOC1
    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //set to last SOC that is converted and it 
will set INT1 flag ADCA1
    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1; //enable INT1 flag
    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is cleared
    EDIS;

    // Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
    // which is connected to CPU-Timer 1, and CPU int 14, which is connected
    // to CPU-Timer 2:  int 12 is for the SWI.
    IER |= M_INT1;
    IER |= M_INT6;
    IER |= M_INT8;  // SCIC SCID
    IER |= M_INT9;  // SCIA  CANB
    IER |= M_INT12;
    IER |= M_INT13;
    IER |= M_INT14;

    // Enable TINT0 in the PIE: Group 1 interrupt 7
    PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
    // Enable SWI in the PIE: Group 12 interrupt 9
    PieCtrlRegs.PIEIER12.bit.INTx9 = 1;
    PieCtrlRegs.PIEIER6.bit.INTx3 = 1;  //SPiB
    // ----- code for CAN start here -----
    // Enable CANB in the PIE: Group 9 interrupt 7
    PieCtrlRegs.PIEIER9.bit.INTx7 = 1;
    // ----- code for CAN end here -----

    //JS Enable ADCA1 in the PIE: Group 1 interrupt 1
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;

    // ----- code for CAN start here -----
    // Enable the CAN interrupt signal
    CanbRegs.CAN_GLB_INT_EN.bit.GLBINT0_EN = 1;
    // ----- code for CAN end here -----

    init_serialSCIC(&SerialC,115200);
    init_serialSCID(&SerialD,115200);
    init_eQEPs();
    // Enable global Interrupts and higher priority real-time debug events
    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global realtime interrupt DBGM
    // ----- code for CAN start here -----

    //    // Transmit Message
    //    // Initialize the transmit message object used for sending CAN messages.
    //    // Message Object Parameters:
    //    //      Message Object ID Number: 0
    //    //      Message Identifier: 0x1
    //    //      Message Frame: Standard
```

```
//    //      Message Type: Transmit
//    //      Message ID Mask: 0x0
//    //      Message Object Flags: Transmit Interrupt
//    //      Message Data Length: 4 Bytes
//    //
//    CANsetupMessageObject(CANB_BASE, TX_MSG_OBJ_ID, 0x1, CAN_MSG_FRAME_STD,
//                          CAN_MSG_OBJ_TYPE_TX, 0, CAN_MSG_OBJ_TX_INT_ENABLE,
//                          TX_MSG_DATA_LENGTH);

// Measured Distance from 1
// Initialize the receive message object 1 used for receiving CAN messages.
// Message Object Parameters:
//      Message Object ID Number: 1
//      Message Identifier: 0x060b0101
//      Message Frame: Standard
//      Message Type: Receive
//      Message ID Mask: 0x0
//      Message Object Flags: Receive Interrupt
//      Message Data Length: 8 Bytes (Note that DLC field is a "don't care"
//      for a Receive mailbox)
//
CANsetupMessageObject(CANB_BASE, RX_MSG_OBJ_ID_1, 0x060b0101, CAN_MSG_FRAME_EXT,
                      CAN_MSG_OBJ_TYPE_RX, 0, CAN_MSG_OBJ_RX_INT_ENABLE,
                      RX_MSG_DATA_LENGTH);

// Measured Distance from 2
// Initialize the receive message object 2 used for receiving CAN messages.
// Message Object Parameters:
//      Message Object ID Number: 2
//      Message Identifier: 0x060b0102
//      Message Frame: Standard
//      Message Type: Receive
//      Message ID Mask: 0x0
//      Message Object Flags: Receive Interrupt
//      Message Data Length: 8 Bytes (Note that DLC field is a "don't care"
//      for a Receive mailbox)
//

CANsetupMessageObject(CANB_BASE, RX_MSG_OBJ_ID_2, 0x060b0103, CAN_MSG_FRAME_EXT,
                      CAN_MSG_OBJ_TYPE_RX, 0, CAN_MSG_OBJ_RX_INT_ENABLE,
                      RX_MSG_DATA_LENGTH);

// Measurement Quality from 1
// Initialize the receive message object 2 used for receiving CAN messages.
// Message Object Parameters:
//      Message Object ID Number: 3
//      Message Identifier: 0x060b0201
//      Message Frame: Standard
//      Message Type: Receive
//      Message ID Mask: 0x0
//      Message Object Flags: Receive Interrupt
//      Message Data Length: 8 Bytes (Note that DLC field is a "don't care"
//      for a Receive mailbox)
//
```

```
    CANsetupMessageObject(CANB_BASE, RX_MSG_OBJ_ID_3, 0x060b0201, CAN_MSG_FRAME_EXT,
                          CAN_MSG_OBJ_TYPE_RX, 0, CAN_MSG_OBJ_RX_INT_ENABLE,
                          RX_MSG_DATA_LENGTH);

    // Measurement Quality from 2
    // Initialize the receive message object 2 used for receiving CAN messages.
    // Message Object Parameters:
    //      Message Object ID Number: 4
    //      Message Identifier: 0x060b0202
    //      Message Frame: Standard
    //      Message Type: Receive
    //      Message ID Mask: 0x0
    //      Message Object Flags: Receive Interrupt
    //      Message Data Length: 8 Bytes (Note that DLC field is a "don't care"
    //      for a Receive mailbox)
    //

    CANsetupMessageObject(CANB_BASE, RX_MSG_OBJ_ID_4, 0x060b0203, CAN_MSG_FRAME_EXT,
                          CAN_MSG_OBJ_TYPE_RX, 0, CAN_MSG_OBJ_RX_INT_ENABLE,
                          RX_MSG_DATA_LENGTH);

    //
    // Start CAN module operations
    //
    CanbRegs.CAN_CTL.bit.Init = 0;
    CanbRegs.CAN_CTL.bit.CCE = 0;

    //    // Initialize the transmit message object data buffer to be sent
    //    txMsgData[0] = 0x12;
    //    txMsgData[1] = 0x34;
    //    txMsgData[2] = 0x56;
    //    txMsgData[3] = 0x78;


    //    // Loop Forever - A message will be sent once per second.
    //    for(;;)
    //    {
    //
    //        CANsendMessage(CANB_BASE, TX_MSG_OBJ_ID, TX_MSG_DATA_LENGTH,
txMsgData);
    //        txMsgCount++;
    //        DEVICE_DELAY_US(1000000);
    //    }


    // ----- code for CAN end here -----


    // IDLE loop. Just sit and loop forever (optional):
    while(1)
    {
        if (UARTPrint == 1 ) {
            //serial_printf(&SerialA,"Num Timer2:%ld Num SerialRX:
%ld\r\n",CpuTimer2.InterruptCount,numRXA);
            //JS lab 7 exercise1.7, print values to test
```

```
            //                   serial_printf(&SerialA,"a:%.3f,%.3f,%.3f
g:%.3f,%.3f,%.3f\r\n",accelx,accely,accelz,gyrox,gyroy,gyroz);
            //                   serial_printf(&SerialA,"D1 %ld D2 %ld",dis_1,dis_3);
            //            serial_printf(&SerialA," St1 %ld St2
%ld\n\r",measure_status_1,measure_status_3);
            serial_printf(&SerialA,"tilt value:%.3f gyro value:%.3f left wheel:%.3f
right wheel:%.3f\n\r", tilt_value, gyro_value, LeftWheel, RightWheel);

            UARTPrint = 0;
        }
    }
}


// SWI_isr,  Using this interrupt as a Software started interrupt
__interrupt void SWI_isr(void) {

    // These three lines of code allow SWI_isr, to be interrupted by other interrupt
functions
    // making it lower priority than all other Hardware interrupts.
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
    asm("       NOP");                    // Wait one cycle
    EINT;                                 // Clear INTM to enable interrupts


    //JS copy from the guideline to communicate with Labview
    if (NewLVData == 1) {
        NewLVData = 0;
        Segbot_refspeed = fromLVvalues[0];
        turnrate = fromLVvalues[1];
        printLV3 = fromLVvalues[2];
        printLV4 = fromLVvalues[3];
        printLV5 = fromLVvalues[4];
        printLV6 = fromLVvalues[5];
        printLV7 = fromLVvalues[6];
        printLV8 = fromLVvalues[7];
    }
    if((numSWIcalls%62) == 0) { // change to the counter variable of you selected
4ms. timer
        DataToLabView.floatData[0] = x;
        DataToLabView.floatData[1] = y;
        DataToLabView.floatData[2] = bearing;
        DataToLabView.floatData[3] = 2.0*((float)numTimer0calls)*.001;
        DataToLabView.floatData[4] = 3.0*((float)numTimer0calls)*.001;
        DataToLabView.floatData[5] = (float)numTimer0calls;
        DataToLabView.floatData[6] = (float)numTimer0calls*4.0;
        DataToLabView.floatData[7] = (float)numTimer0calls*5.0;
        LVsenddata[0] = '*'; // header for LVdata
        LVsenddata[1] = '$';
        for (int i=0;i<LVNUM_TOFROM_FLOATS*4;i++) {
            if (i%2==0) {
                LVsenddata[i+2] = DataToLabView.rawData[i/2] & 0xFF;
            } else {
                LVsenddata[i+2] = (DataToLabView.rawData[i/2]>>8) & 0xFF;
            }
```

```
        }
        serial_sendSCID(&SerialD, LVsenddata, 4*LVNUM_TOFROM_FLOATS + 2);
    }
    if ((numSWIcalls%50) == 0) {
        PieCtrlRegs.PIEIFR12.bit.INTx9 = 1;  // Manually cause the interrupt for the
SWI
    }

    //JS angular position of wheels
    LeftWheel = -readEncLeft();
    RightWheel = readEncRight();
    //JS convert to linear position , pos = angle*radius
    LeftWheelmeter = LeftWheel*radius;
    RightWheelmeter = RightWheel*radius;

    //JS calculate the velocity of wheels using change position over changing time
    PosLeft_k = LeftWheelmeter;
    VLeftK = (PosLeft_k-PosLeft_k_1)/0.004;
    PosLeft_k_1 = PosLeft_k;
    PosRt_k = RightWheelmeter;
    VRtK = (PosRt_k-PosRt_k_1)/0.004;
    PosRt_k_1 = PosRt_k;

    //    //JS implemented pose calculations
    angvel_left = VLeftK/radius;
    angvel_rt = VRtK/radius;
    bearing = (radius/widthrob)*(RightWheel-LeftWheel);
    beavg = 0.5*(RightWheel+LeftWheel);
    beavgdot = 0.5*(angvel_left+angvel_rt);
    xrdot = radius*beavgdot*cos(bearing);
    yrdot = radius*beavgdot*sin(bearing);
    //JS x and y calculated using Trapezoidal Rule integration
    x = x_1+(0.004*(xrdot+x_1dot)/2);
    y = y_1+(0.004*(yrdot+y_1dot)/2);
    x_1 = x;
    y_1 = y;

    // Insert SWI ISR Code here.......
    //JS lab7 exercise 3, solve for the difference equation of the discrete transfer
function
    vel_Left =0.6*vel_Left_pre +100*LeftWheel-100*ltwheel_pre;
    vel_Right =0.6*vel_Right_pre +100*RightWheel-100*rtwheel_pre;
    //JS lab7 exercise 3, solve derivative of gyro_value
    gyrorate_dot=0.6*gyroratedot_pre+100*gyro_value-100*gyrovalue_pre;


    vel_Left_pre=vel_Left;
    vel_Right_pre =vel_Right;
    ltwheel_pre=LeftWheel;
    rtwheel_pre =RightWheel;
    gyroratedot_pre=gyrorate_dot;
    gyrovalue_pre=gyro_value;
    //JS lab 7 exercise4,difference between two wheels' angle
    whldiff = LeftWheel-RightWheel;
```

```
    //JS lab 7 exercise4, solve for velocity using filter equations for discrete
transfer function ((166.667z-166.667)/(z-0.333)) at 0.004s period
    vel_whldiff = 0.333*vel_whldiff_1+166.67*whldiff-166.67*whldiff_1;
    //JS lab 7 exercise4, create variables for old values
    vel_whldiff_1=vel_whldiff;
    whldiff_1=whldiff;

    //JS lab 7 exercise4, use trapezoidal rule to calculate turnref
    turnref = turnref_1+(0.004*(turnrate+turnrate_1)/2);
    turnref_1=turnref;
    turnrate_1=turnrate;
    //JS lab 7 exercise4, error between turnref and feedback signal
    errordiff =turnref-whldiff;
    //JS lab 7 exercise4, integrate errordiff using trapezoidal rule
    intdiff = intdiff_1+(0.004*(errordiff+errordiff_1)/2);
    intdiff_1 =intdiff;
    errordiff_1 =errordiff;
    //JS lab 7 exercise4, PID control turn command
    turn = Kp*errordiff+Ki*intdiff-Kd*vel_whldiff;

    //JS lab 7 exercise5, calculate error between Segbot_refspeed and average wheel
velocity
    avgwheelvel=(vel_Left+vel_Right)/2.0;
    espeed=Segbot_refspeed-avgwheelvel;
    //JS lab 7 exercise5, calculate IK_espeed
    IK_espeed = IK_espeed_1+(0.004*(espeed+espeed_1)/2.0);
    //JS lab 7 exercise5, implement PI speed control
    forwardbackwardcommand=kpspeed*espeed+kispeed*IK_espeed;

    //JS lab 7 exercise4, guard against integral windup
    if (fabs(turn)>3.0){
        intdiff=intdiff_1;
    }


    //Js lab 7 exercise4, satuarate turn between -4 and 4
    if (turn>4.0){
        turn = 4.0;
    }
    if (turn<-4.0){
        turn = -4.0;
    }

    //JS lab 7 exercise5, guard against integral espeed
    if (fabs(forwardbackwardcommand)>3){
        IK_espeed =IK_espeed_1;
    }
    //JS lab 7 exercise5, saturate forwardbackwardcommand
    if (forwardbackwardcommand>4.0){
        forwardbackwardcommand =4.0;
    }
    if (forwardbackwardcommand<-4.0){
        forwardbackwardcommand =-4.0;
    }
```

```
//JS lab7 exercise3, calculate the control law
    ubal =-k1*tilt_value-k2*gyro_value-k3*avgwheelvel-k4*gyrorate_dot;
    //JS lab 7 exercise3, 4,5, calculate control effort of motors
    uleft=ubal/2+turn-forwardbackwardcommand;
    uright=ubal/2-turn-forwardbackwardcommand;
    //JS lab 7 exercise3, drive motors with control efforts
    setEPWM2A(uright);
    setEPWM2B(-uleft);
    numSWIcalls++;

    DINT;

}

// cpu_timer0_isr - CPU Timer0 ISR
__interrupt void cpu_timer0_isr(void)
{
    CpuTimer0.InterruptCount++;

    numTimer0calls++;
    //JS copy from the guideline to communicate with Labview
    //    if (NewLVData == 1) {
    //        NewLVData = 0;
    //        vref = fromLVvalues[0];
    //        turn = fromLVvalues[1];
    //        printLV3 = fromLVvalues[2];
    //        printLV4 = fromLVvalues[3];
    //        printLV5 = fromLVvalues[4];
    //        printLV6 = fromLVvalues[5];
    //        printLV7 = fromLVvalues[6];
    //        printLV8 = fromLVvalues[7];
    //    }
    //    if((numTimer0calls%62) == 0) { // change to the counter variable of you
selected 4ms. timer
    //        DataToLabView.floatData[0] = x;
    //        DataToLabView.floatData[1] = y;
    //        DataToLabView.floatData[2] = bearing;
    //        DataToLabView.floatData[3] = 2.0*((float)numTimer0calls)*.001;
    //        DataToLabView.floatData[4] = 3.0*((float)numTimer0calls)*.001;
    //        DataToLabView.floatData[5] = (float)numTimer0calls;
    //        DataToLabView.floatData[6] = (float)numTimer0calls*4.0;
    //        DataToLabView.floatData[7] = (float)numTimer0calls*5.0;
    //        LVsenddata[0] = '*'; // header for LVdata
    //        LVsenddata[1] = '$';
    //        for (int i=0;i<LVNUM_TOFROM_FLOATS*4;i++) {
    //            if (i%2==0) {
    //                LVsenddata[i+2] = DataToLabView.rawData[i/2] & 0xFF;
    //            } else {
    //                LVsenddata[i+2] = (DataToLabView.rawData[i/2]>>8) & 0xFF;
    //            }
    //        }
    //        serial_sendSCID(&SerialD, LVsenddata, 4*LVNUM_TOFROM_FLOATS + 2);
    //    }
    //    if ((numTimer0calls%50) == 0) {
```

```
    //        PieCtrlRegs.PIEIFR12.bit.INTx9 = 1;  // Manually cause the interrupt
for the SWI
    //    }


    //    //JS angular position of wheels
    //    LeftWheel = -readEncLeft();
    //    RightWheel = readEncRight();
    //    //JS convert to linear position , pos = angle*radius
    //    LeftWheelmeter = LeftWheel*radius;
    //    RightWheelmeter = RightWheel*radius;
    //
    //    //JS calculate the velocity of wheels using change position over changing
time
    //    PosLeft_k = LeftWheelmeter;
    //    VLeftK = (PosLeft_k-PosLeft_k_1)/0.004;
    //    PosLeft_k_1 = PosLeft_k;
    //    PosRt_k = RightWheelmeter;
    //    VRtK = (PosRt_k-PosRt_k_1)/0.004;
    //    PosRt_k_1 = PosRt_k;
    //
    //    //JS implemented coupled PI controller structure for left
    //    eturn = turn+(VLeftK - VRtK);
    //    eleftk = vref - VLeftK-kpturn*eturn;
    //    if (fabs(uleft)>10.0){
    //       Ileftk = Ileftk_1*0.95;
    //    } else {
    //        Ileftk = Ileftk_1 + (0.004*(eleftk+eleftk_1)/2);
    //    }
    //
    //    uleft = kp*eleftk+Ki*Ileftk;
    //    eleftk_1 = eleftk;
    //    Ileftk_1 = Ileftk;
    //
    //    //JS implemented coupled PI controller structure for right
    //    ertk = vref - VRtK+kpturn*eturn;
    //    if (fabs(uleft)>10.0){
    //       Irtk = Irtk_1*0.95;
    //    } else {
    //        Irtk = Irtk_1 + (0.004*(ertk+ertk_1)/2);
    //    }
    //
    //    uright = kp*ertk+Ki*Irtk;
    //    ertk_1 = ertk;
    //    Irtk_1 = Irtk;
    //
    //    //JS pass calculated us to setEPWM to run the motors
    ////    setEPWM2A(uright);
    ////    setEPWM2B(-uleft);
    //
    //    //JS implemented pose calculations
    //    angvel_left = VLeftK/radius;
    //    angvel_rt = VRtK/radius;
    //    bearing = (radius/widthrob)*(RightWheel-LeftWheel);
    //    beavg = 0.5*(RightWheel+LeftWheel);
```

```
    //    beavgdot = 0.5*(angvel_left+angvel_rt);
    //    xrdot = radius*beavgdot*cos(bearing);
    //    yrdot = radius*beavgdot*sin(bearing);
    //    //JS x and y calculated using Trapezoidal Rule integration
    //    x = x_1+(0.004*(xrdot+x_1dot)/2);
    //    y = y_1+(0.004*(yrdot+y_1dot)/2);
    //    x_1 = x;
    //    y_1 = y;
    //
    //    //JS copy from guideline
    //    if (measure_status_1 == 0) {
    //        distright = dis_1;
    //    } else {
    //        distright = 1400; // set to max reading if error
    //    }
    //    if (measure_status_3 == 0) {
    //        distfront = dis_3;
    //    } else {
    //        distfront = 1400; // set to max reading if error
    //    }
    //
    //
    //    //JS right wall following controller
    //    if (rtwallfollow==1){
    //        turn = kprt *(refrt-distright);
    //        vref=0.25;
    //        if (distfront <threshold1){
    //            rtwallfollow =0;
    //        }
    //        //JS when microphone filter signal is around 2000Hz, robot holds
position
    //        if (ykb1>0.4 || ykb1<-0.4){
    //            vref=-0.25;
    //        }
    //    }
    //    else{
    //        turn = kpft *(refft-distfront);
    //        vref=0.25;
    //        if(distfront>threshold2){
    //            rtwallfollow =1;
    //        }
    //        //JS when microphone filter signal is around 2000Hz, robot holds
position
    //        if (ykb1>0.4 || ykb1<-0.4){
    //            vref=-0.25;
    //        }
    //    }
    //
    //    distright_1 =distright;
    //    distfront_1 = distfront;
    //
    //
    //    if ((numTimer0calls%250) == 0) {
    //        displayLEDletter(LEDdisplaynum);
    //        LEDdisplaynum++;
```

```
    //        if (LEDdisplaynum == 0xFFFF) {  // prevent roll over exception
    //            LEDdisplaynum = 0;
    //        }
    //    }


    //Clear GPIO9 Low to act as a Slave Select. Right now, just to scope. Later to
select DAN28027 chip
    //GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;
    //    SpibRegs.SPIFFRX.bit.RXFFIL = 2; // Issue the SPIB_RX_INT when two values
are in the RX FIFO
    //    SpibRegs.SPITXBUF = 0x4A3B; // 0x4A3B and 0xB517 have no special meaning.
Wanted to send
    //    SpibRegs.SPITXBUF = 0xB517; // something so you can see the pattern on the
Oscilloscope
    //    SpibRegs.SPIFFRX.bit.RXFFIL = 3; // Issue the SPIB_RX_INT when two values
are in the RX FIFO
    //    SpibRegs.SPITXBUF = 0xDA;
    //    SpibRegs.SPITXBUF = 500; // PWM value
    //    SpibRegs.SPITXBUF = 2200; // PWM Value


    //
    //    if ((numTimer0calls%50) == 0) {
    //        // Blink LaunchPad Red LED
    //        GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
    //    }
    //
    //
    //    // Acknowledge this interrupt to receive more interrupts from group 1
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}


// cpu_timer1_isr - CPU Timer1 ISR
__interrupt void cpu_timer1_isr(void)
{

    CpuTimer1.InterruptCount++;
}

// cpu_timer2_isr CPU Timer2 ISR
__interrupt void cpu_timer2_isr(void)
{
    //    // Blink LaunchPad Blue LED
    //    GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;
    //
    //    CpuTimer2.InterruptCount++;
    //
    //    if ((CpuTimer2.InterruptCount % 10) == 0) {
    //        //          UARTPrint = 1;
    //    }
}
```

```
void setupSpib(void) //Call this function in main() somewhere after the DINT; line of
code.
{
    int16_t temp = 0;
    //Step 1.
    // cut and paste here all the SpibRegs initializations you found for part 3. Make
sure the TXdelay in between each transfer to 0. Also donÃ¢â‚¬â„¢t forget to cut and
paste the GPIO settings for GPIO9, 63, 64, 65, 66 which are also a part of the SPIB
setup.
    GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0); // Set as GPIO9 and used as DAN28027 SS
    GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL); // Make GPIO9 an Output Pin
    GpioDataRegs.GPASET.bit.GPIO9 = 1; //Initially Set GPIO9/SS High so DAN28027 is
not selected

    GPIO_SetupPinMux(66, GPIO_MUX_CPU1, 0); // Set as GPIO66 and used as MPU-9250 SS
    GPIO_SetupPinOptions(66, GPIO_OUTPUT, GPIO_PUSHPULL); // Make GPIO66 an Output
Pin
    GpioDataRegs.GPCSET.bit.GPIO66 = 1; //Initially Set GPIO66/SS High so MPU-9250 is
not selected

    GPIO_SetupPinMux(63, GPIO_MUX_CPU1, 15); //Set GPIO63 pin to SPISIMOB
    GPIO_SetupPinMux(64, GPIO_MUX_CPU1, 15); //Set GPIO64 pin to SPISOMIB
    GPIO_SetupPinMux(65, GPIO_MUX_CPU1, 15); //Set GPIO65 pin to SPICLKB

    EALLOW;
    GpioCtrlRegs.GPBPUD.bit.GPIO63 = 0; // Enable Pull-ups on SPI PINs Recommended by
TI for SPI Pins
    GpioCtrlRegs.GPCPUD.bit.GPIO64 = 0;
    GpioCtrlRegs.GPCPUD.bit.GPIO65 = 0;
    GpioCtrlRegs.GPBQSEL2.bit.GPIO63 = 3; // Set I/O pin to asynchronous mode
recommended for SPI
    GpioCtrlRegs.GPCQSEL1.bit.GPIO64 = 3; // Set I/O pin to asynchronous mode
recommended for SPI
    GpioCtrlRegs.GPCQSEL1.bit.GPIO65 = 3; // Set I/O pin to asynchronous mode
recommended for SPI
    EDIS;

    // ---------------------------------------------------------------------------
    SpibRegs.SPICCR.bit.SPISWRESET = 0; // Put SPI in Reset

    SpibRegs.SPICTL.bit.CLK_PHASE = 1; //This happens to be the mode for both the
DAN28027 and
    SpibRegs.SPICCR.bit.CLKPOLARITY = 0; //The MPU-9250, Mode 01.
    SpibRegs.SPICTL.bit.MASTER_SLAVE = 1; // Set to SPI Master
    SpibRegs.SPICCR.bit.SPICHAR = 0xF; // Set to transmit and receive 16-bits each
write to SPITXBUF
    SpibRegs.SPICTL.bit.TALK = 1; // Enable transmission
    SpibRegs.SPIPRI.bit.FREE = 1; // Free run, continue SPI operation
    SpibRegs.SPICTL.bit.SPIINTENA = 0; // Disables the SPI interrupt

    SpibRegs.SPIBRR.bit.SPI_BIT_RATE = 49; // Set SCLK bit rate to 1 MHz so 1us
period. SPI base clock is
    // 50MHZ. And this setting divides that base clock to create SCLKÃ¢â‚¬â„¢s period
    SpibRegs.SPISTS.all = 0x0000; // Clear status flags just in case they are set for
some reason
```

```
    SpibRegs.SPIFFTX.bit.SPIRST = 1;// Pull SPI FIFO out of reset, SPI FIFO can
resume transmit or receive.
    SpibRegs.SPIFFTX.bit.SPIFFENA = 1; // Enable SPI FIFO enhancements
    SpibRegs.SPIFFTX.bit.TXFIFO = 0; // Write 0 to reset the FIFO pointer to zero,
and hold in reset
    SpibRegs.SPIFFTX.bit.TXFFINTCLR = 1; // Write 1 to clear SPIFFTX[TXFFINT] flag
just in case it is set

    SpibRegs.SPIFFRX.bit.RXFIFORESET = 0; // Write 0 to reset the FIFO pointer to
zero, and hold in reset
    SpibRegs.SPIFFRX.bit.RXFFOVFCLR = 1; // Write 1 to clear SPIFFRX[RXFFOVF] just in
case it is set
    SpibRegs.SPIFFRX.bit.RXFFINTCLR = 1; // Write 1 to clear SPIFFRX[RXFFINT] flag
just in case it is set
    SpibRegs.SPIFFRX.bit.RXFFIENA = 1; // Enable the RX FIFO Interrupt. RXFFST >=
RXFFIL

    //SpibRegs.SPIFFCT.bit.TXDLY = 16; //Set delay between transmits to 16 spi
clocks. Needed by DAN28027 chip
    SpibRegs.SPIFFCT.bit.TXDLY = 0;

    SpibRegs.SPICCR.bit.SPISWRESET = 1; // Pull the SPI out of reset

    SpibRegs.SPIFFTX.bit.TXFIFO = 1; // Release transmit FIFO from reset.
    SpibRegs.SPIFFRX.bit.RXFIFORESET = 1; // Re-enable receive FIFO operation
    SpibRegs.SPICTL.bit.SPIINTENA = 1; // Enables SPI interrupt. !! I dont think this
is needed. Need to Test

    SpibRegs.SPIFFRX.bit.RXFFIL =16; //Interrupt Level to 16 words or more received
into FIFO causes interrupt. This is just the initial setting for the register. Will
be changed below

    //-----------------------------------------------------------------------------
----------------------------------

    //Step 2.
    // perform a multiple 16-bit transfer to initialize MPU-9250 registers
0x13,0x14,0x15,0x16
    // 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C 0x1D, 0x1E, 0x1F. Use only one SS low to
high for all these writes
    // some code is given, most you have to fill you yourself.
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1; // Slave Select Low


    // Perform the number of needed writes to SPITXBUF to write to all 13 registers.
Remember we are sending 16-bit transfers, so two registers at a time after the first
16-bit transfer.
    // To address 00x13 write 0x00
    SpibRegs.SPITXBUF = (0x1300 | 0x0000);
    // To address 00x14 write 0x00
    // To address 00x15 write 0x00
    SpibRegs.SPITXBUF = (0x0000 | 0x0000);
    // To address 00x16 write 0x00
    // To address 00x17 write 0x00
```

```
    SpibRegs.SPITXBUF = (0x0000 | 0x0000);
    // To address 00x18 write 0x00
    // To address 00x19 write 0x13
    SpibRegs.SPITXBUF = (0x0000 | 0x0013);
    // To address 00x1A write 0x02
    // To address 00x1B write 0x00
    SpibRegs.SPITXBUF = (0x0200 | 0x0000);
    // To address 00x1C write 0x08
    // To address 00x1D write 0x06
    SpibRegs.SPITXBUF = (0x0800 | 0x0006);
    // To address 00x1E write 0x00
    // To address 00x1F write 0x00
    SpibRegs.SPITXBUF = (0x0000 | 0x0000);

    // wait for the correct number of 16-bit values to be received into the RX FIFO
    while(SpibRegs.SPIFFRX.bit.RXFFST !=7);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Slave Select High
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    // ???? read the additional number of garbage receive values off the RX FIFO to
clear out the RX FIFO
    DELAY_US(10); // Delay 10us to allow time for the MPU-2950 to get ready for next
transfer.

    //Step 3.
    // perform a multiple 16-bit transfer to initialize MPU-9250 registers
0x23,0x24,0x25,0x26
    // 0x27, 0x28, 0x29. Use only one SS low to high for all these writes
    // some code is given, most you have to fill you yourself.
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1; // Slave Select Low

    // Perform the number of needed writes to SPITXBUF to write to all 7 registers
    // To address 00x23 write 0x00
    SpibRegs.SPITXBUF = (0x2300 | 0x0000);
    // To address 00x24 write 0x40
    // To address 00x25 write 0x8C
    SpibRegs.SPITXBUF = (0x4000 | 0x008C);
    // To address 00x26 write 0x02
    // To address 00x27 write 0x88
    SpibRegs.SPITXBUF = (0x0200 | 0x0088);
    // To address 00x28 write 0x0C
    // To address 00x29 write 0x0A
    SpibRegs.SPITXBUF = (0x0C00 | 0x000A);

    // wait for the correct number of 16-bit values to be received into the RX FIFO
    while(SpibRegs.SPIFFRX.bit.RXFFST !=4);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Slave Select High
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
    temp = SpibRegs.SPIRXBUF;
```

```
    temp = SpibRegs.SPIRXBUF;
    // ???? read the additional number of garbage receive values off the RX FIFO to
clear out the RX FIFO
    DELAY_US(10); // Delay 10us to allow time for the MPU-2950 to get ready for next
transfer.

    //Step 4.
    // perform a single 16-bit transfer to initialize MPU-9250 register 0x2A
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    // Write to address 0x2A the value 0x81
    SpibRegs.SPITXBUF = (0x2A00 | 0x0081);
    // wait for one byte to be received
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);

    // The Remainder of this code is given to you and you do not need to make any
changes.
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x3800 | 0x0001); // 0x3800
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x3A00 | 0x0001); // 0x3A00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x6400 | 0x0001); // 0x6400
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x6700 | 0x0003); // 0x6700
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x6A00 | 0x0020); // 0x6A00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x6B00 | 0x0001); // 0x6B00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
```

```
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x7500 | 0x0071); // 0x7500
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    //JS change to calibrate SPI
    SpibRegs.SPITXBUF = (0x7700 | 0x00E9); // 0x7700
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    //JS change to calibrate SPI
    SpibRegs.SPITXBUF = (0x7800 | 0x00AE); // 0x7800
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    //JS change to calibrate SPI
    SpibRegs.SPITXBUF = (0x7A00 | 0x0017); // 0x7A00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    //JS change to calibrate SPI
    SpibRegs.SPITXBUF = (0x7B00 | 0x00EA); // 0x7B00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x7D00 | 0x0019); // 0x7D00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(10);
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
    SpibRegs.SPITXBUF = (0x7E00 | 0x0082); // 0x7E00
    while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    temp = SpibRegs.SPIRXBUF;
    DELAY_US(50);

    // Clear SPIB interrupt source just in case it was issued due to any of the above
initializations.
    SpibRegs.SPIFFRX.bit.RXFFOVFCLR=1; // Clear Overflow flag
    SpibRegs.SPIFFRX.bit.RXFFINTCLR=1; // Clear Interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;
}

int16_t spivalue1 = 0;
```

```c
int16_t spivalue2 = 0;
int16_t spivalue3 = 0;
__interrupt void SPIB_isr(void) {
    GpioDataRegs.GPCSET.bit.GPIO66 = 1;
    spivalue1 = SpibRegs.SPIRXBUF;
    accelx_raw = SpibRegs.SPIRXBUF;
    accely_raw = SpibRegs.SPIRXBUF;
    accelz_raw = SpibRegs.SPIRXBUF;
    spivalue2 = SpibRegs.SPIRXBUF;
    gyrox_raw = SpibRegs.SPIRXBUF;
    gyroy_raw = SpibRegs.SPIRXBUF;
    gyroz_raw = SpibRegs.SPIRXBUF;
//JS lab7 exercise 1.6, covert to units of g and degree/s
    accelx = accelx_raw*(4.0/32767.0);
    accely = accely_raw*(4.0/32767.0);
    accelz = accelz_raw*(4.0/32767.0);

    gyrox = gyrox_raw*(250.0/32767.0);
    gyroy = gyroy_raw*(250.0/32767.0);
    gyroz = gyroz_raw*(250.0/32767.0);

    //JS lab7 exercise2, use kalman filter to produce tilt measurement
    //Code to be copied into SPIB_ISR interrupt function after the IMU measurements
have been collected.
    if(calibration_state == 0){
        calibration_count++;
        if (calibration_count == 2000) {
            calibration_state = 1;
            calibration_count = 0;
        }
    } else if(calibration_state == 1) {
        accelx_offset+=accelx;
        accely_offset+=accely;
        accelz_offset+=accelz;
        gyrox_offset+=gyrox;
        gyroy_offset+=gyroy;
        gyroz_offset+=gyroz;
        calibration_count++;
        if (calibration_count == 2000) {
            calibration_state = 2;
            accelx_offset/=2000.0;
            accely_offset/=2000.0;
            accelz_offset/=2000.0;
            gyrox_offset/=2000.0;
            gyroy_offset/=2000.0;
            gyroz_offset/=2000.0;
            calibration_count = 0;
            doneCal = 1;
        }
    } else if(calibration_state == 2) {
        accelx -=(accelx_offset);
        accely -=(accely_offset);
        accelz -=(accelz_offset-accelzBalancePoint);
        gyrox -= gyrox_offset;
        gyroy -= gyroy_offset;
```

```
        gyroz -= gyroz_offset;
        /*--------------Kalman Filtering code start-----------------------------------
-----------------
    ------------------*/
        // Prediction Step
        float tiltrate = (gyrox*PI)/180.0; // rad/s
        float pred_tilt, z, y, S;
        pred_tilt = kalman_tilt + T*tiltrate;
        pred_P = kalman_P + Q;
        // Update Step
        z = -accelz; // Note the negative here due to the polarity of AccelZ
        y = z - pred_tilt;
        S = pred_P + R;
        kalman_K = pred_P/S;
        kalman_tilt = pred_tilt + kalman_K*y;
        kalman_P = (1 - kalman_K)*pred_P;
        SpibNumCalls++;
        // Kalman Filter used
        tilt_array[SpibNumCalls] = kalman_tilt;
        gyro_array[SpibNumCalls] = tiltrate;
        LeftWheelArray[SpibNumCalls] = -readEncLeft();
        RightWheelArray[SpibNumCalls] = readEncRight();
        if (SpibNumCalls >= 3) { // should never be greater than 3
            tilt_value = (tilt_array[0] + tilt_array[1] + tilt_array[2] +
tilt_array[3])/4.0;
            gyro_value = (gyro_array[0] + gyro_array[1] + gyro_array[2] +
gyro_array[3])/4.0;

LeftWheel=(LeftWheelArray[0]+LeftWheelArray[1]+LeftWheelArray[2]+LeftWheelArray[3])/4
.0;

RightWheel=(RightWheelArray[0]+RightWheelArray[1]+RightWheelArray[2]+RightWheelArray[
3])/4.0;
            SpibNumCalls = -1;
            PieCtrlRegs.PIEIFR12.bit.INTx9 = 1; // Manually cause the interrupt for
the SWI

        }
    }
    timecount++;
    if((timecount%200) == 0) {
        if(doneCal == 0) {
            GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1; // Blink Blue LED while
calibrating
        }
        GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Always Block Red LED
        UARTPrint = 1; // Tell While loop to print
    }

//JS lab7 exercise1.7, call functions to send motor angles
    //    ltangle = -readEncLeft();
    //    rtangle = readEncRight();
    //JS lab7 exercise1.8, drive motors open-loop
    //    setEPWM2A(uright);
    //    setEPWM2B(-uleft);
```

```
    SpibRegs.SPIFFRX.bit.RXFFOVFCLR=1;  // Clear Overflow flag
    SpibRegs.SPIFFRX.bit.RXFFINTCLR=1;  // Clear Interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;

}

// ----- code for CAN start here -----
__interrupt void can_isr(void)
{
    int i = 0;

    uint32_t status;

    GpioDataRegs.GPBSET.bit.GPIO52 = 1;
    //
    // Read the CAN interrupt status to find the cause of the interrupt
    //
    status = CANgetInterruptCause(CANB_BASE);

    //
    // If the cause is a controller status interrupt, then get the status
    //
    if(status == CAN_INT_INT0ID_STATUS)
    {
        //
        // Read the controller status.  This will return a field of status
        // error bits that can indicate various errors.  Error processing
        // is not done in this example for simplicity.  Refer to the
        // API documentation for details about the error status bits.
        // The act of reading this status will clear the interrupt.
        //
        status = CANgetStatus(CANB_BASE);

    }

    //
    // Check if the cause is the transmit message object 1
    //
    //    else if(status == TX_MSG_OBJ_ID)
    //    {
    //        //
    //        // Getting to this point means that the TX interrupt occurred on
    //        // message object 1, and the message TX is complete.  Clear the
    //        // message object interrupt.
    //        //
    //        CANclearInterruptStatus(CANB_BASE, TX_MSG_OBJ_ID);
    //
    //        //
    //        // Since the message was sent, clear any error flags.
    //        //
    //        errorFlag = 0;
    //    }
```

```
//
// Check if the cause is the receive message object 2
//
else if(status == RX_MSG_OBJ_ID_1)
{
    //
    // Get the received message
    //
    CANreadMessage(CANB_BASE, RX_MSG_OBJ_ID_1, rxMsgData);

    for(i = 0; i<2; i++)
    {
        dis_raw_1[i] = rxMsgData[i];
    }

    dis_1 = 256*dis_raw_1[1] + dis_raw_1[0];

    measure_status_1 = rxMsgData[2];

    //
    // Getting to this point means that the RX interrupt occurred on
    // message object 2, and the message RX is complete.  Clear the
    // message object interrupt.
    //
    CANclearInterruptStatus(CANB_BASE, RX_MSG_OBJ_ID_1);

    //
    // Increment a counter to keep track of how many messages have been
    // received. In a real application this could be used to set flags to
    // indicate when a message is received.
    //
    rxMsgCount_1++;

    //
    // Since the message was received, clear any error flags.
    //
    errorFlag = 0;
    GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
}

else if(status == RX_MSG_OBJ_ID_2)
{
    //
    // Get the received message
    //
    CANreadMessage(CANB_BASE, RX_MSG_OBJ_ID_2, rxMsgData);

    for(i = 0; i<2; i++)
    {
        dis_raw_3[i] = rxMsgData[i];
    }

    dis_3 = 256*dis_raw_3[1] + dis_raw_3[0];
```

```
        measure_status_3 = rxMsgData[2];

        //
        // Getting to this point means that the RX interrupt occurred on
        // message object 2, and the message RX is complete.  Clear the
        // message object interrupt.
        //
        CANclearInterruptStatus(CANB_BASE, RX_MSG_OBJ_ID_2);

        //
        // Increment a counter to keep track of how many messages have been
        // received. In a real application this could be used to set flags to
        // indicate when a message is received.
        //
        rxMsgCount_3++;

        //
        // Since the message was received, clear any error flags.
        //
        errorFlag = 0;
        GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
    }

    else if(status == RX_MSG_OBJ_ID_3)
    {
        //
        // Get the received message
        //
        CANreadMessage(CANB_BASE, RX_MSG_OBJ_ID_3, rxMsgData);

        for(i = 0; i<4; i++)
        {
            lightlevel_raw_1[i] = rxMsgData[i];
            quality_raw_1[i] = rxMsgData[i+4];

        }

        lightlevel_1 = ((256.0*256.0*256.0)*lightlevel_raw_1[3] +
(256.0*256.0)*lightlevel_raw_1[2] + 256.0*lightlevel_raw_1[1] +
lightlevel_raw_1[0])/65535;
        quality_1 = ((256.0*256.0*256.0)*quality_raw_1[3] +
(256.0*256.0)*quality_raw_1[2] + 256.0*quality_raw_1[1] + quality_raw_1[0])/65535;


        //
        // Getting to this point means that the RX interrupt occurred on
        // message object 2, and the message RX is complete.  Clear the
        // message object interrupt.
        //
        CANclearInterruptStatus(CANB_BASE, RX_MSG_OBJ_ID_3);

        //
        // Since the message was received, clear any error flags.
        //
        errorFlag = 0;
```

```
        GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
    }


    else if(status == RX_MSG_OBJ_ID_4)
    {
        //
        // Get the received message
        //
        CANreadMessage(CANB_BASE, RX_MSG_OBJ_ID_4, rxMsgData);

        for(i = 0; i<4; i++)
        {
            lightlevel_raw_3[i] = rxMsgData[i];
            quality_raw_3[i] = rxMsgData[i+4];

        }

        lightlevel_3 = ((256.0*256.0*256.0)*lightlevel_raw_3[3] +
(256.0*256.0)*lightlevel_raw_3[2] + 256.0*lightlevel_raw_3[1] +
lightlevel_raw_3[0])/65535;
        quality_3 = ((256.0*256.0*256.0)*quality_raw_3[3] +
(256.0*256.0)*quality_raw_3[2] + 256.0*quality_raw_3[1] + quality_raw_3[0])/65535;

        //
        // Getting to this point means that the RX interrupt occurred on
        // message object 2, and the message RX is complete.  Clear the
        // message object interrupt.
        //
        CANclearInterruptStatus(CANB_BASE, RX_MSG_OBJ_ID_4);

        //
        // Since the message was received, clear any error flags.
        //
        errorFlag = 0;
        GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
    }



    //
    // If something unexpected caused the interrupt, this would handle it.
    //
    else
    {
        //
        // Spurious interrupt handling can go here.
        //
    }

    //
    // Clear the global interrupt flag for the CAN interrupt line
    //
    CANclearGlobalInterruptStatus(CANB_BASE, CAN_GLOBAL_INT_CANINT0);
```

```
    //
    // Acknowledge this interrupt located in group 9
    //
    InterruptclearACKGroup(INTERRUPT_ACK_GROUP9);
}
// ----- code for CAN end here -----
```