Final Project: Self-balanced Segbot, Position Tracking & Auto Moving

Name: Yifan Yang

NetID: yifany8

1. Introduction

The goal of this project is to implement kinematic algorithms on the self-balanced Segbot to find its position on X-Y plane while it is moving and implement auto moving function. In the process of this project, I added kinematics equations and auto moving function to the code from ME 461 Lab 6, and calculated the specific coordinate position of on the X-Y plane during the movement of the Segbot using turning angle of two wheels. Finally, the Segbot can automatically move to any position within the allowable range of space and connecting lines. The calculated location of the Segbot can be displayed on Tera Term. Error of the position is within the acceptable error range.

2. Hardware Equipment

The Segbot is mainly composed of self-designed circuit board (green board), TMS320F28379D (red board), DC motors (JGA25-370-CE), DC power supply, power wheels, support wheel and holders.

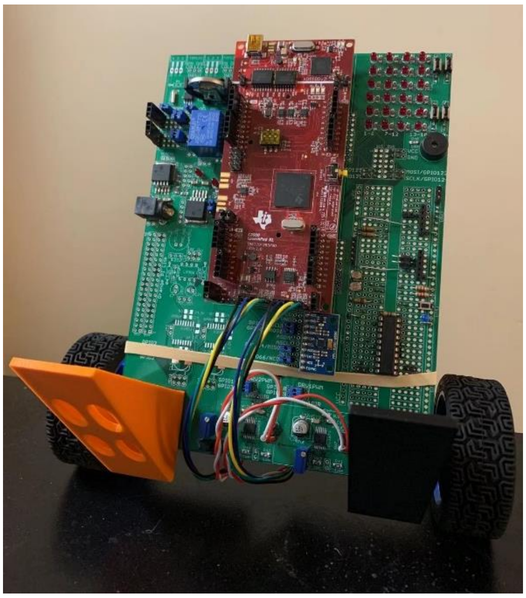

Figure 1 self-balanced Segbot

3. Method

The main code of the project is in Appendix. Here in the Method part, only the important part of the code will be explained.

Below is the code of variable declaration of kinematic equations. xm and ym are the position of the Segbot on the X-Y plane which we would like to know. Rw is the radius of the wheel. To avoid the error of directly measuring of wheel radius using ruler, the radius is obtained by calculating the difference of actual turning angle of left and right wheels before and after the Segbot moving forward for 1 foot distance. This extra experiment is finished before in Lab 4 section. theta_left and theta_right is used to make it possible to reset the LeftWheel and RightWheel values. xT and yT are the target postion, you can enter any position you want. phi_T is the difference between the direction of target and body yaw angle. stage is the stage value at which the Segbot is at in the auto moving part. When stage is 0, the Segbot is prepare stage, it will calculate the difference between the direction of target and its body yaw angle. When stage is 1, the Segbot will start turning in a given angular velocity until it is facing the target. When stage is 2, the Segbot will move towards the target until it is closed to the target point. When stage is 3, the Segbot stop moving and return to normal stage.

```
// (Final Project) position tracking variables
int stage = 0; // moving stage
float xm = 0.0; // m, Wheel axle midpoint position in X axis
float ym = 0.0; // m, Wheel axle midpoint position in Y axis
float xT = 0.0; // m, target position in X axis
float yT = 0.0; // m, target position in Y axis
float xm_dot = 0.0; // m/s, Midpoint velocity on X axis
float ym_dot = 0.0; // m/s, Midpoint velocity on Y axis
float xm_dot_1 = 0.0;
float ym_dot_1 = 0.0;
float Rw = ((0.3048/9.431778)+(0.3048/9.342178))/2.0; // m, Wheel radius (9.9.431778 and 9.342178 are the Rads per ft of the wheels, measured in Lab4)
float theta_left = 0.0; // Rad, Left wheel angle (make it possible to reset LeftWheel and RightWheel)
float theta_right = 0.0; // Rad, Right wheel angle
float theta_dot = 0.0; // Rad/s, Average turning speed of wheels
float theta = 0.0; // Rad, Average angle of left and right wheel
float theta_1 = 0.0;
float phi = 0.0; // Rad, Body yaw angle
float phi_T = 0.0; // Rad, direction from position now to target
float W = 0.182; // m, Distance between the centers of the two wheels
```

```
int reset = 0; // Reset flag for reset position to zero (also set LeftWheel and RightWheel to zero)
```

Below is the code of initialization of EPwm5, which is used to trigger ADCA channels 2 and 3 every 1 millisecond.

```
//EPWM5
EALLOW;
EPwm5Regs.ETSEL.bit.SOCAEN = 0; // Disable SOC on A group
EPwm5Regs.TBCTL.bit.CTRMODE = 3; // freeze counter
EPwm5Regs.ETSEL.bit.SOCASEL = 2; // Select Event when counter equal to PRD
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
//ME461, Lab 3 Lab 5, Page 3 of 11
EPwm5Regs.TBPRD = 50000; // Set Period to 1ms sample. Input clock is 50MHz.
// 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
EPwm5Regs.TBCTL.bit.CTRMODE = 0; //unfreeze, and enter up count mode
EDIS;
```

Below is the code of initialization of ADCs. In this code, only ADCA is initialized. The ADCs are used to read the feedback signal of sensors and actuators that are used on the Segbot.

```
    //write configurations for all ADCs ADCA, ADCB, ADCC, ADCD
    AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
```

```
    AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    //Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    //power up the ADCs
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    //delay for 1ms to allow ADC time to power up
    DELAY_US(1000);
    //Select the channels to convert and end of conversion flag
    //Many statements commented out, To be used when using ADCA or ADCB
    //ME461, Lab 3 Lab 5, Page 4 of 11

    //ADCA
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2; //SOC0 will convert Channel you choose Does not have to be A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK cycles = 75ns
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0xD;// EPWM5 ADCSOCA or another trigger you choose will trigger SOC0

    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3; //SOC1 will convert Channel you choose Does not have to be A1
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK cycles = 75ns
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0xD;// EPWM5 ADCSOCA or another trigger you choose will trigger 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;

    EALLOW;
    DacaRegs.DACOUTEN.bit.DACOUTEN = 1;//enable dacA output-->uses ADCINA0
```

```
    DacaRegs.DACCTL.bit.LOADMODE = 0;//load on next sysclk
    DacaRegs.DACCTL.bit.DACREFSEL = 1;//use ADC VREF as reference voltage
    DacbRegs.DACOUTEN.bit.DACOUTEN = 1;//enable dacB output-->uses ADCINA1
    DacbRegs.DACCTL.bit.LOADMODE = 0;//load on next sysclk
    DacbRegs.DACCTL.bit.DACREFSEL = 1;//use ADC VREF as reference voltage
    EDIS;
```

Below is the code of initialization of EPwm6. EPwm6A and EPwm6B are used here to control the turning speed of the two DC motors. Here EPwm6Regs.CMPA.bit.CMPA and EPwm6Regs.CMPB.bit.CMPB are set to zero so that the Segbot will not move in the first few seconds when the code start running.

```
  //initialize the PWM register 6A, 6B
    EPwm6Regs.TBCTL.bit.CTRMODE = 0;
    EPwm6Regs.TBCTL.bit.FREE_SOFT = 2;
    EPwm6Regs.TBCTL.bit.PHSEN = 0;
    EPwm6Regs.TBCTL.bit.CLKDIV = 0;

    EPwm6Regs.TBCTR = 0;
    EPwm6Regs.TBPRD = 2500;
    EPwm6Regs.TBPHS.bit.TBPHS = 0;
      EPwm6Regs.CMPA.bit.CMPA = 0;
      EPwm6Regs.AQCTLA.bit.CAU = 1;
      EPwm6Regs.AQCTLA.bit.ZRO = 2;

      EPwm6Regs.CMPB.bit.CMPB = 0;
      EPwm6Regs.AQCTLB.bit.CBU = 1;
      EPwm6Regs.AQCTLB.bit.ZRO = 2;
```

Below is the code of SWI_isr function, which is the most important part that implement the balancing control law and kinematic equations. Ubal = -K1*tilt – K2*gyrorate – K3*(velLeft + velRight)/2 is the control law we used here. Ubal is the control effort for both motors. In the X-Y position tracking part, another two variable, theta_left and theta_right are used to show the left and right angle of the wheels and also enable reset function. These two values are set to zero so that the current position of Segbot is set to (0,0) and the x axis and y axis are reset (the positive direction of x-axis is the front of the Segbot, and the positive direction of y axis is the left of the Segbot.

```
// 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

     // Insert SWI ISR Code here.......

     turnref = turnref_1 + 0.002 * (turnrate + turnrate_1);

     WhlDiff = LeftWheel - RightWheel;
     vel_WhlDiff = 0.333 * vel_WhlDiff_1 + 2.0/(3.0*0.004)*(WhlDiff - WhlDiff_1);
     eDiff = turnref - WhlDiff;
     intDiff = intDiff_1 + 0.002 * (eDiff + eDiff_1);

     turn = Kp*eDiff + Ki*intDiff - Kd*vel_WhlDiff;

     if(fabs(turn)>3){
         intDiff = intDiff_1;
     }
     if(turn > 4.0){
         turn = 4.0;
     }
     if(turn < -4.0){
         turn = -4.0;
     }

     WhlDiff_1 = WhlDiff;
     vel_WhlDiff_1 = vel_WhlDiff;
     eDiff_1 =  eDiff;
     intDiff_1 = intDiff;
     turnref_1 = turnref;
     turnrate_1 = turnrate;

     v_l = 0.6 * v_l_1 + 100 * (LeftWheel - LeftWheel_1);
     v_r = 0.6 * v_r_1 + 100 * (RightWheel - RightWheel_1);

```

```
    Ubal = -K1 * tilt_value - K2 * gyro_value - K3 * (v_l +
v_r)/2.0;

    v_l_1 = v_l;
    v_r_1 = v_r;

    uLeft = (Ubal/2.0) + turn + FwdBackOffset;
    uRight = (Ubal/2.0) - turn + FwdBackOffset;

    setEPWM6A(uLeft);
    setEPWM6B(-uRight);

    // Final Project add code here
    if (reset == 1) {
        xm = 0.0;
        ym = 0.0;
        xm_dot = 0.0;
        ym_dot = 0.0;
        xm_dot_1 = 0.0;
        ym_dot_1 = 0.0;
        theta_dot = 0.0;
        theta = 0.0;
        theta_1 = 0.0;
        phi = 0.0;
        theta_left = 0.0;
        theta_right =0.0;
        reset = 0;
    }
    theta_left = theta_left + (LeftWheel - LeftWheel_1);
    theta_right = theta_right + (RightWheel - RightWheel_1);
    theta = 0.5 * (theta_left + theta_right);
    phi = Rw / W * (theta_right - theta_left);
    theta_dot = (theta - theta_1) / 0.004;
    xm_dot = Rw * theta_dot * cos(phi);
    ym_dot = Rw * theta_dot * sin(phi);
    xm = xm + 0.004 * (xm_dot + xm_dot_1);
    ym = ym + 0.004 * (ym_dot + ym_dot_1);

    theta_1 = theta;
    xm_dot = xm_dot_1;
    ym_dot = ym_dot_1;
    LeftWheel_1 = LeftWheel;
    RightWheel_1 = RightWheel;

```

```
    // Final project auto move part
    if (stage == 0) {    // prepare stage
        phi_T = atan2(yT-ym,xT-xm);  // specify the direction towards the target
    }
    if (stage == 1) {    // turning stage
        if (fabs(phi_T - phi) < 0.015) { // if the error between phi and phi_T is small enough, stop turning and go to stage 2
            turnrate = 0.0;
            stage = 2;
        }
        else if (phi_T > 0) {
            turnrate = -1.0; // turn left to face the target
        }
        else if (phi_T < 0) {
            turnrate = 1.0; // turn right to face the target
        }
    }
    if (stage == 2) {    // moving stage
        if ((fabs(xm-xT) < 0.01) && (fabs(ym-yT) < 0.01)) {
            turnrate = 0.0;
            FwdBackOffset = 0.0;
            stage = 3;
        }
        else {
            FwdBackOffset = -1.5;
        }
    }
    numSWIcalls++;
    DINT;
}
```

Below is the code of SPIB_ISR function. When the SPI is finished transmitting, which also means it is done receiving, the SPIB_ISR function will be called by the hardware. Inside SPIB_ISR interrupt function, MPU-9250's Slave Select will be pulled high and the 6IMU values will be read (gyro_x, gyro_y, gyro_z, accel_x, accel_y, accel_z). to reduce error, Kalman filter is used.

```
__interrupt void SPIB_isr(void){

    int16_t temp1 = 0;
    numSPIcalls++;

```

```
    temp1 = SpibRegs.SPIRXBUF;
    x_accel = SpibRegs.SPIRXBUF;
    y_accel = SpibRegs.SPIRXBUF;
    z_accel = SpibRegs.SPIRXBUF;
    temp1 = SpibRegs.SPIRXBUF;
    x_gyro = SpibRegs.SPIRXBUF;
    y_gyro = SpibRegs.SPIRXBUF;
    z_gyro = SpibRegs.SPIRXBUF;
    //
    GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Set GPIO 66 to end Slave Select of MPU.

    gyrox = x_gyro * 250.0 / 32767.0;
    gyroy = y_gyro * 250.0 / 32767.0;
    gyroz = y_gyro * 250.0 / 32767.0;
    accelx = x_accel * 4.0 / 32767.0;
    accely = y_accel * 4.0 / 32767.0;
    accelz = z_accel * 4.0 / 32767.0;

    //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---------------------------------------------------------------------*/
        float tiltrate = (gyrox*PI)/180.0; // rad/s
        float pred_tilt, z, y, S;

        tilt_rate = tiltrate;

        // Prediction Step
        pred_tilt = kalman_tilt + T*tiltrate;
        pred_P = kalman_P + Q;

        // Update Step
        z = -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
    }

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

Below is the code of serialRXA function. This function is used to receive command (char) over UARTA. Keys “WASD” are used to control movement of the segbot. Key “R” is used to reset the position value and X-Y axis.

```
// This function is called each time a char is recieved over UARTA.
//use WASD to control sebot moving
void serialRXA(serial_t *s, char data) {
    numRXA ++;
    if (data == 'a') {                          // turn left
        turnrate = turnrate - 0.2;
    } else if (data == 'd') {                   // turn right
        turnrate = turnrate + 0.2;
```

```
    } else if (data == 'w') {                    // go forward
        FwdBackOffset = FwdBackOffset - 0.2;
    } else if (data == 's') {                    // go backward
        FwdBackOffset = FwdBackOffset + 0.2;
    } else if (data == 'r') {                    // reset position
        turnrate = 0;
        FwdBackOffset = 0;
        reset = 1;
    } else {                                     // reset speed
        turnrate = 0;
        FwdBackOffset = 0;
    }
}
```

Below is the code of the functions of readEncLeft and readEncRight. These two functions are used to read the angles of left and right wheels. There are 20 north south magnet poles in the encoder disk, so during every turn of the wheel, 20 square waves will be received. The quadrature decoder mode multiplies this by 4 so there is 80 counts per one rev of the DC motors. The motor’s gear ratio is 18.7, so the angle of the wheel is raw_value *(2*PI/80.0)/18.7.

```
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
    // 20 North South magnet poles in the encoder disk so 20 square waves per one revolution of the
    // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so 80 counts per one rev
    // of the DC motor's back shaft. Then the gear motor's gear ratio is 18.7.
    return (raw*(2*PI/80.0)/18.7);
}

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
```

```
    // 20 North South magnet poles in the encoder disk so 20 square waves per one revolution of the
    // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so 80 counts per one rev
    // of the DC motor's back shaft. Then the gear motor's gear ratio is 18.7.
    return (raw*(2*PI/80.0)/18.7);
}
```

Below is the code of setEPWM6A and setEPWM6B functions. These two functions are used to control the turning speed of the two wheels. 2500 is the value of TBPRD, which is the maximum value of CMPA or CMPB.

```
void setEPWM6A (float controleffort){

   if (controleffort >= 10){
       controleffort = 10;
   }
   else if (controleffort <= -10){
       controleffort = -10;
   }

    if (controleffort >= 0){
        GpioDataRegs.GPASET.bit.GPIO29 = 1;
    }
    else{
        GpioDataRegs.GPACLEAR.bit.GPIO29 = 1;
    }
    EPwm6Regs.CMPA.bit.CMPA = 2500 * (fabs(controleffort) / 10.0);
}

void setEPWM6B (float controleffort){

    if (controleffort >= 10){
        controleffort = 10;

    }
    else if (controleffort <= -10){
        controleffort = -10;
    }

    if (controleffort >= 0){
        GpioDataRegs.GPBSET.bit.GPIO32 = 1;
```

```
    }
    else{
        GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;
    }
    EPwm6Regs.CMPB.bit.CMPB = 2500 * (fabs(controleffort) / 10.0);
}
```

4. Result

After running the main code, manually keep the Segbot at its balancing position for about 2~3 second. The wheels will not spin at the first 2~3 seconds, and then start spinning and go to its balancing pose. Press ‘R’ in Tera Term, x and y values will be reset to zero. Then press ‘W’ make the self-balanced segbot to go forward, press ‘S’ to go backward, press ‘A’ to turn left, press ‘D’ to turn right and press any other keys to stop. The segbot successfully moves in accordance with the button commands while maintaining balance. X and Y coordinates are displayed in Tera Term. Manually measure the position and compare it with calculation result. The Error is in within the acceptable error range. The code can successfully track the position of Segbot on the X-Y plane. For auto moving part, you can set (xY,yT) to any position you want. press 'T' to start moving, the segbot will first turn to the direction of target and then move towards the target till it reach the target. There will be some error between target position and actual position mainly because the connecting line will pull the Segbot from its original balancing point. If you want to enter a second target point. Set value 'stage' to be 0 before you press 'T'.

5. Conclusion

In conclusion, the code can successfully control the Segbot to move according to the button commands, automatically move to the target point you want while maintaining balance and can also track the position of Segbot on the X-Y plane.

6. Appendix

Below is the main code of the project:

```
// 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"

#define PI          3.1415926535897932384626433832795
#define TWOPI       6.283185307179586476925286766559
#define HALFPI      1.5707963267948966192313216916398
#define g          9.8

// 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);
__interrupt void ADCA_ISR (void);
__interrupt void SPIB_isr(void);

void serialRXA(serial_t *s, char data);
void setupSpib(void);
void init_eQEPs(void);
float readEncLeft(void);
float readEncRight(void);
void setEPWM6A (float);
void setEPWM6B (float);

// Count variables
uint32_t numTimer0calls = 0;
uint32_t numSWIcalls = 0;
uint32_t numRXA = 0;
uint16_t UARTPrint = 0;
uint32_t numADCAcalls = 0;
uint32_t numSPIcalls = 0;

float out1 = 0.0;
float out2 = 0.0;
float analogvalue0 = 0.0;
float analogvalue1 = 0.0;
float adcinput1;
float adcinput2;
float LeftWheel = 0.0;
float RightWheel = 0.0;
float LeftWheel_1 = 0.0;
float RightWheel_1 = 0.0;
```

```
float LW_factor = 9.8;
float RW_factor = 9.8;
float uLeft = 0.0;
float uRight = 0.0;
float XLeft_K = 0.0;
float XRight_K = 0.0;
float XLeft_K_1 = 0.0;
float XRight_K_1 = 0.0;
float VLeft_K = 0.0;
float VRight_K = 0.0;
float e_K_L = 0.0;
float e_K_R = 0.0;
float e_K_1_L = 0.0;
float e_K_1_R = 0.0;
float I_K_L = 0.0;
float I_K_R = 0.0;
float I_K_1_L = 0.0;
float I_K_1_R = 0.0;
float Vref = 1;
float Kpt = 3.0;
float e_turn = 0.0;

int32_t digital0 = 0;
int32_t digital1 = 0;

int16_t x_gyro = 0;
int16_t y_gyro = 0;
int16_t z_gyro = 0;
int16_t x_accel = 0;
int16_t y_accel = 0;
int16_t z_accel = 0;

float gyrox = 0.0;
float gyroy = 0.0;
float gyroz = 0.0;
float accelx = 0.0;
float accely = 0.0;
float accelz = 0.0;

// 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 = -.762;
int16 IMU_data[9];
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};

//balance control
float Ubal = 0.0;
float tilt_rate = 0.0;
float v_l = 0.0;
float v_r = 0.0;
float v_l_1 = 0.0;
float v_r_1 = 0.0;

float K1 = -30.0;
float K2 = -2.8;
float K3 = -1.0;

//turn control
float WhlDiff = 0.0;
float WhlDiff_1 = 0.0;
float vel_WhlDiff = 0.0;
float vel_WhlDiff_1 = 0.0;
float intDiff = 0.0;
float intDiff_1 = 0.0;
float eDiff = 0.0;
float eDiff_1 = 0.0;

float turnref = 0.0;
float turnref_1 = 0.0;
float turn = 0.0;
float turnrate = 0.0;
float turnrate_1 = 0.0;
float FwdBackOffset = 0.0;

float Kp = 3.0;
float Ki = 20.0;
float Kd = 0.08;
```

```

// Kalman Filter
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;

// (Final Project) position tracking variables
int stage = 0; // moving stage
float xm = 0.0; // m, Wheel axle midpoint position in X axis
float ym = 0.0; // m, Wheel axle midpoint position in Y axis
float xT = 0.0; // m, target position in X axis
float yT = 0.0; // m, target position in Y axis
float xm_dot = 0.0; // m/s, Midpoint velocity on X axis
float ym_dot = 0.0; // m/s, Midpoint velocity on Y axis
float xm_dot_1 = 0.0;
float ym_dot_1 = 0.0;
float Rw = ((0.3048/9.431778)+(0.3048/9.342178))/2.0; // m, Wheel radius (9.9.431778 and 9.342178 are the Rads per ft of the wheels, measured in Lab4)
float theta_left = 0.0; // Rad, Left wheel angle (make it possible to reset LeftWheel and RightWheel)
float theta_right = 0.0; // Rad, Right wheel angle
float theta_dot = 0.0; // Rad/s, Average turning speed of wheels
float theta = 0.0; // Rad, Average angle of left and right wheel
float theta_1 = 0.0;
float phi = 0.0; // Rad, Body yaw angle
float phi_T = 0.0; // Rad, direction from position now to target
float W = 0.182; // m, Distance between the centers of the two wheels
int reset = 0; // Reset flag for reset position to zero (also set LeftWheel and RightWheel to zero)

void setDACA(float dacouta0) {
    if (dacouta0 > 3.0) dacouta0 = 3.0;
    if (dacouta0 < 0.0) dacouta0 = 0.0;
```

```
    DacaRegs.DACVALS.bit.DACVALS = 4095.0 / 3.0 * dacouta0; // perform scaling of 0-3 to 0-4095
}

void setDACB(float dacouta1) {
    if (dacouta1 > 3.0) dacouta1 = 3.0;
    if (dacouta1 < 0.0) dacouta1 = 0.0;
    DacbRegs.DACVALS.bit.DACVALS = 4095.0 / 3.0 * dacouta1; // perform scaling of 0-3 to 0-4095
}

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

    InitGpio();

    // Blue LED on LuanchPad
    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(52, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(52, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;

    // LED3
    GPIO_SetupPinMux(67, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(67, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPCCLEAR.bit.GPIO67 = 1;

    // LED4
```

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

    // LED5
    GPIO_SetupPinMux(95, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(95, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPCCLEAR.bit.GPIO95 = 1;

    // LED6
    GPIO_SetupPinMux(97, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(97, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPDCLEAR.bit.GPIO97 = 1;

    // LED7
    GPIO_SetupPinMux(111, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(111, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPDCLEAR.bit.GPIO111 = 1;

    // LED8
    GPIO_SetupPinMux(130, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(130, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPECLEAR.bit.GPIO130 = 1;

    // LED9
    GPIO_SetupPinMux(131, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(131, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPECLEAR.bit.GPIO131 = 1;

    // LED10
    GPIO_SetupPinMux(4, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(4, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;

    // LED11
    GPIO_SetupPinMux(5, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(5, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;

    // LED12
    GPIO_SetupPinMux(6, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(6, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;

```

```
    // LED13
    GPIO_SetupPinMux(7, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(7, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;

    // LED14
    GPIO_SetupPinMux(8, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(8, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;

    // LED15
    GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;

    // LED16
    GPIO_SetupPinMux(24, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(24, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO24 = 1;

    // LED17
    GPIO_SetupPinMux(25, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(25, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;

    // LED18
    GPIO_SetupPinMux(26, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(26, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO26 = 1;

    // LED19
    GPIO_SetupPinMux(27, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(27, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;

    // LED20
    GPIO_SetupPinMux(60, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(60, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1;

    // LED21
    GPIO_SetupPinMux(61, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(61, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;
```

```

    // LED22
    GPIO_SetupPinMux(157, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(157, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPECLEAR.bit.GPIO157 = 1;

    // LED23
    GPIO_SetupPinMux(158, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(158, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPECLEAR.bit.GPIO158 = 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;

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

    //PushButton 1
    GPIO_SetupPinMux(122, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(122, GPIO_INPUT, GPIO_PULLUP);

    //PushButton 2
    GPIO_SetupPinMux(123, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(123, GPIO_INPUT, GPIO_PULLUP);

    //PushButton 3
    GPIO_SetupPinMux(124, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(124, GPIO_INPUT, GPIO_PULLUP);

    //PushButton 4
    GPIO_SetupPinMux(125, GPIO_MUX_CPU1, 0);
    GPIO_SetupPinOptions(125, GPIO_INPUT, GPIO_PULLUP);

    // control right motor input by PWM
    GPIO_SetupPinMux(10, GPIO_MUX_CPU1, 1);
```

```
    GPIO_SetupPinOptions(10, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;

    // control left motor input by PWM
    GPIO_SetupPinMux(11, GPIO_MUX_CPU1, 1);
    GPIO_SetupPinOptions(11, GPIO_OUTPUT, GPIO_PUSHPULL);
    GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;

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

    setupSpib();
    init_eQEPs();

    //disable pull-up resistor
    EALLOW;  // Below are protected registers
    GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;
    GpioCtrlRegs.GPAPUD.bit.GPIO22 = 1;
    GpioCtrlRegs.GPEPUD.bit.GPIO159 = 1;
    EDIS;

    // 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.SCIC_RX_INT = &RXCINT_recv_ready;
    PieVectTable.SCID_RX_INT = &RXDINT_recv_ready;
    PieVectTable.SCIA_TX_INT = &TXAINT_data_sent;
    PieVectTable.SCIC_TX_INT = &TXCINT_data_sent;
    PieVectTable.SCID_TX_INT = &TXDINT_data_sent;
    PieVectTable.ADCA1_INT = &ADCA_ISR;
    PieVectTable.SPIB_RX_INT = &SPIB_isr;
    PieVectTable.EMIF_ERROR_INT = &SWI_isr;
    EDIS;    // This is needed to disable write to EALLOW protected registers

    //EPWM5
      EALLOW;
      EPwm5Regs.ETSEL.bit.SOCAEN = 0; // Disable SOC on A group
      EPwm5Regs.TBCTL.bit.CTRMODE = 3; // freeze counter
      EPwm5Regs.ETSEL.bit.SOCASEL = 2; // Select Event when counter equal to PRD
      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
      //ME461, Lab 3 Lab 5, Page 3 of 11
      EPwm5Regs.TBPRD = 50000; // Set Period to 1ms sample. Input clock is 50MHz.
      // 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
      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
```

```
    AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
    AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT, ADC_SIGNALMODE_SINGLE); //read calibration settings
    //Set pulse positions to late
    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;
    //power up the ADCs
    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;
    //delay for 1ms to allow ADC time to power up
    DELAY_US(1000);
    //Select the channels to convert and end of conversion flag
    //Many statements commented out, To be used when using ADCA or ADCB
    //ME461, Lab 3 Lab 5, Page 4 of 11

    //ADCA
    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2; //SOC0 will convert Channel you choose Does not have to be A0
    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK cycles = 75ns
    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0xD;// EPWM5 ADCSOCA or another trigger you choose will trigger SOC0

    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3; //SOC1 will convert Channel you choose Does not have to be A1
    AdcaRegs.ADCSOC1CTL.bit.ACQPS = 14; //sample window is acqps + 1 SYSCLK cycles = 75ns
    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0xD;// EPWM5 ADCSOCA or another trigger you choose will trigger 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;

    EALLOW;
    DacaRegs.DACOUTEN.bit.DACOUTEN = 1;//enable dacA output-->uses ADCINA0
    DacaRegs.DACCTL.bit.LOADMODE = 0;//load on next sysclk
    DacaRegs.DACCTL.bit.DACREFSEL = 1;//use ADC VREF as reference voltage
    DacbRegs.DACOUTEN.bit.DACOUTEN = 1;//enable dacB output-->uses ADCINA1
    DacbRegs.DACCTL.bit.LOADMODE = 0;//load on next sysclk
    DacbRegs.DACCTL.bit.DACREFSEL = 1;//use ADC VREF as reference voltage
    EDIS;

    // 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 second:
    // 200MHz CPU Freq, 1 second Period (in uSeconds)
    ConfigCpuTimer(&CpuTimer0, 200, 1000);
    ConfigCpuTimer(&CpuTimer1, 200, 20000);
    ConfigCpuTimer(&CpuTimer2, 200, 4000);

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

    init_serial(&SerialA,115200,serialRXA);
    //    init_serial(&SerialC,115200,serialRXC);
    //    init_serial(&SerialD,115200,serialRXD);

    // 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_INT8;  // SCIC SCID
    IER |= M_INT9;  // SCIA
    IER |= M_INT12;
    IER |= M_INT13;
    IER |= M_INT14;
    IER |= M_INT6;  //SPIB_RX

    //Enable ADCA1 in the PIE: Group 1 interrupt 1
    PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
    // Enable SPIB_RX: Group 6 interrupt 3
    PieCtrlRegs.PIEIER6.bit.INTx3 = 1;
    // 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;

    //initialize the PWM register 6A, 6B
      EPwm6Regs.TBCTL.bit.CTRMODE = 0;
      EPwm6Regs.TBCTL.bit.FREE_SOFT = 2;
      EPwm6Regs.TBCTL.bit.PHSEN = 0;
      EPwm6Regs.TBCTL.bit.CLKDIV = 0;

      EPwm6Regs.TBCTR = 0;
      EPwm6Regs.TBPRD = 2500;
      EPwm6Regs.TBPHS.bit.TBPHS = 0;
      EPwm6Regs.CMPA.bit.CMPA = 0;
      EPwm6Regs.AQCTLA.bit.CAU = 1;
      EPwm6Regs.AQCTLA.bit.ZRO = 2;

      EPwm6Regs.CMPB.bit.CMPB = 0;
      EPwm6Regs.AQCTLB.bit.CBU = 1;
      EPwm6Regs.AQCTLB.bit.ZRO = 2;

    // Enable global Interrupts and higher priority real-time debug events
    EINT;  // Enable Global interrupt INTM
    ERTM;  // Enable Global realtime interrupt DBGM

    // IDLE loop. Just sit and loop forever (optional):
    while(1)
    {
        if (UARTPrint == 1 ) {
```

```
            serial_printf(&SerialA, "LeftWheel: %.3f, RightWheel: %.3f, theta_left: %.3f, theta_right: %.3f, X: %.3f, Y: %.3f \r\n",LeftWheel,RightWheel,theta_left,theta_right,xm,ym);
            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

    // Insert SWI ISR Code here.......

    turnref = turnref_1 + 0.002 * (turnrate + turnrate_1);

    WhlDiff = LeftWheel - RightWheel;
    vel_WhlDiff = 0.333 * vel_WhlDiff_1 + 2.0/(3.0*0.004)*(WhlDiff - WhlDiff_1);
    eDiff = turnref - WhlDiff;
    intDiff = intDiff_1 + 0.002 * (eDiff + eDiff_1);

    turn = Kp*eDiff + Ki*intDiff - Kd*vel_WhlDiff;

    if(fabs(turn)>3){
        intDiff = intDiff_1;
    }
    if(turn > 4.0){
        turn = 4.0;
    }
    if(turn < -4.0){
        turn = -4.0;
    }

    WhlDiff_1 = WhlDiff;
    vel_WhlDiff_1 = vel_WhlDiff;
```

```
    eDiff_1 =  eDiff;
    intDiff_1 = intDiff;
    turnref_1 = turnref;
    turnrate_1 = turnrate;

    v_l = 0.6 * v_l_1 + 100 * (LeftWheel - LeftWheel_1);
    v_r = 0.6 * v_r_1 + 100 * (RightWheel - RightWheel_1);

    Ubal = -K1 * tilt_value - K2 * gyro_value - K3 * (v_l + v_r)/2.0;

    v_l_1 = v_l;
    v_r_1 = v_r;

    uLeft = (Ubal/2.0) + turn + FwdBackOffset;
    uRight = (Ubal/2.0) - turn + FwdBackOffset;

    setEPWM6A(uLeft);
    setEPWM6B(-uRight);

    // Final Project add code here
    if (reset == 1) {
        xm = 0.0;
        ym = 0.0;
        xm_dot = 0.0;
        ym_dot = 0.0;
        xm_dot_1 = 0.0;
        ym_dot_1 = 0.0;
        theta_dot = 0.0;
        theta = 0.0;
        theta_1 = 0.0;
        phi = 0.0;
        theta_left = 0.0;
        theta_right =0.0;
        reset = 0;
    }
    theta_left = theta_left + (LeftWheel - LeftWheel_1);
    theta_right = theta_right + (RightWheel - RightWheel_1);
    theta = 0.5 * (theta_left + theta_right);
    phi = Rw / W * (theta_right - theta_left);
    theta_dot = (theta - theta_1) / 0.004;
    xm_dot = Rw * theta_dot * cos(phi);
    ym_dot = Rw * theta_dot * sin(phi);
    xm = xm + 0.004 * (xm_dot + xm_dot_1);
```

```
    ym = ym + 0.004 * (ym_dot + ym_dot_1);

    theta_1 = theta;
    xm_dot = xm_dot_1;
    ym_dot = ym_dot_1;
    LeftWheel_1 = LeftWheel;
    RightWheel_1 = RightWheel;

    // Final project auto move part
    if (stage == 0) {    // prepare stage
        phi_T = atan2(yT-ym,xT-xm);  // specify the direction towards the target
    }
    if (stage == 1) {    // turning stage
        if (fabs(phi_T - phi) < 0.015) { // if the error between phi and phi_T is small enough, stop turning and go to stage 2
            turnrate = 0.0;
            stage = 2;
        }
        else if (phi_T > 0) {
            turnrate = -1.0; // turn left to face the target
        }
        else if (phi_T < 0) {
            turnrate = 1.0; // turn right to face the target
        }
    }
    if (stage == 2) {    // moving stage
        if ((fabs(xm-xT) < 0.01) && (fabs(ym-yT) < 0.01)) {
            turnrate = 0.0;
            FwdBackOffset = 0.0;
            stage = 3;
        }
        else {
            FwdBackOffset = -1.5;
        }
    }

    numSWIcalls++;
    DINT;
}

// cpu_timer0_isr - CPU Timer0 ISR
__interrupt void cpu_timer0_isr(void)
{
```

```
    CpuTimer0.InterruptCount++;

    numTimer0calls++;

//    if ((numTimer0calls%50) == 0) {
//        PieCtrlRegs.PIEIFR12.bit.INTx9 = 1;  // Manually cause the interrupt for the SWI
//    }


    // Blink LaunchPad Red LED
    //GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
    // Blink a number of LEDS
    //    GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;
    //    GpioDataRegs.GPATOGGLE.bit.GPIO8 = 1;
    //    GpioDataRegs.GPATOGGLE.bit.GPIO9 = 1;
    //    GpioDataRegs.GPATOGGLE.bit.GPIO24 = 1;
    //    GpioDataRegs.GPATOGGLE.bit.GPIO25 = 1;
    //    GpioDataRegs.GPATOGGLE.bit.GPIO26 = 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)
{


    // Blink a number of LEDS
    //    GpioDataRegs.GPATOGGLE.bit.GPIO22 = 1;
    //    GpioDataRegs.GPBTOGGLE.bit.GPIO52 = 1;
    //    GpioDataRegs.GPCTOGGLE.bit.GPIO67 = 1;
    //    GpioDataRegs.GPCTOGGLE.bit.GPIO94 = 1;
    //    GpioDataRegs.GPCTOGGLE.bit.GPIO95 = 1;
    //    GpioDataRegs.GPDTOGGLE.bit.GPIO97 = 1;

    CpuTimer1.InterruptCount++;
}

// cpu_timer2_isr CPU Timer2 ISR
__interrupt void cpu_timer2_isr(void)
{
```

```c


    // Blink LaunchPad Blue LED
    //GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;
    // Blink a number of LEDS
    //   GpioDataRegs.GPDTOGGLE.bit.GPIO111 = 1;
    //   GpioDataRegs.GPETOGGLE.bit.GPIO130 = 1;
    //   GpioDataRegs.GPETOGGLE.bit.GPIO131 = 1;
    //   GpioDataRegs.GPATOGGLE.bit.GPIO4 = 1;
    //   GpioDataRegs.GPATOGGLE.bit.GPIO5 = 1;
    //   GpioDataRegs.GPATOGGLE.bit.GPIO6 = 1;
//   CpuTimer2.InterruptCount++;
//      if ((CpuTimer2.InterruptCount % 50) == 0) {
//              UARTPrint = 1;
//      }


}


//b is the filter coefficients
float b[10]={   1.1982297073578186e-02,
             3.2593697188218529e-02,
             8.8809724362308426e-02,
             1.5903360855022139e-01,
             2.0758067282567344e-01,
             2.0758067282567344e-01,
             1.5903360855022139e-01,
             8.8809724362308426e-02,
             3.2593697188218529e-02,
             1.1982297073578186e-02};

//xk is the current ADC reading, xk_1 is the ADC reading one millisecond ago, xk_2 two milliseconds ago, etc
float xk[10];
float zk[10];

float yk = 0.0;
float wk = 0.0;

uint32_t adca0result = 0;
uint32_t adca1result = 0;

//adca1 pie interrupt
```

```c
__interrupt void ADCA_ISR (void)
{
    int i = 0;
    numADCAcalls ++;

    //save adc output
    adca0result = AdcaResultRegs.ADCRESULT0;
    adca1result = AdcaResultRegs.ADCRESULT1;

    // Here covert ADCIND0, ADCIND1 to volts
    analogvalue0 = 3.0*(adca0result / 4095.0);
    analogvalue1 = 3.0*(adca1result / 4095.0);

    xk[0] = analogvalue0;

    for(i=0; i<=9; i++){
        yk += b[i]*xk[i];
    }
    out1 = yk;
    yk = 0.0;

    //Save past states before exiting from the function so that next sample they are the older state
    for(i=9; i>0; i--){
        xk[i] = xk[i-1];
    }

    zk[0] = analogvalue1;

    for(i=0; i<=9; i++){
        wk += b[i]*zk[i];
    }
    out2 = wk;
    wk = 0.0;

    //Save past states before exiting from the function so that next sample they are the older state
    for(i=9; i>0; i--){
        zk[i] = zk[i-1];
    }


    // Code inside CPU Timer 0
    GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
```

```
    SpibRegs.SPIFFRX.bit.RXFFIL = 8;
    SpibRegs.SPITXBUF = ((0x8000)|(0x3A00)); //send random value to the former address
    SpibRegs.SPITXBUF = 0; //to receive x_acceleration
    SpibRegs.SPITXBUF = 0; //to receive y_acceleration
    SpibRegs.SPITXBUF = 0; //to receive z_acceleration
    SpibRegs.SPITXBUF = 0; //receive temp
    SpibRegs.SPITXBUF = 0; //to receive x_gyro
    SpibRegs.SPITXBUF = 0; //to receive y_gyro
    SpibRegs.SPITXBUF = 0; //to receive z_gyro

//    // Print ADCIND0 and ADCIND1's voltage value to TeraTerm every 100ms
//    if ((numADCAcalls % 100) == 0) {
//        UARTPrint = 1;
//    }

    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear interrupt flag
    PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;

}

__interrupt void SPIB_isr(void){

    int16_t temp1 = 0;
    numSPIcalls++;

    temp1 = SpibRegs.SPIRXBUF;
    x_accel = SpibRegs.SPIRXBUF;
    y_accel = SpibRegs.SPIRXBUF;
    z_accel = SpibRegs.SPIRXBUF;
    temp1 = SpibRegs.SPIRXBUF;
    x_gyro = SpibRegs.SPIRXBUF;
    y_gyro = SpibRegs.SPIRXBUF;
    z_gyro = SpibRegs.SPIRXBUF;
    //
    GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Set GPIO 66 to end Slave Select of MPU.

    gyrox = x_gyro * 250.0 / 32767.0;
    gyroy = y_gyro * 250.0 / 32767.0;
    gyroz = y_gyro * 250.0 / 32767.0;
    accelx = x_accel * 4.0 / 32767.0;
    accely = y_accel * 4.0 / 32767.0;
```

```
    accelz = z_accel * 4.0 / 32767.0;


    //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-----------------------------------------------------------------------*/
```

```
        float tiltrate = (gyrox*PI)/180.0; // rad/s
        float pred_tilt, z, y, S;

        tilt_rate = tiltrate;

        // Prediction Step
        pred_tilt = kalman_tilt + T*tiltrate;
        pred_P = kalman_P + Q;

        // Update Step
        z = -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
    }

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

}

// This function is called each time a char is recieved over UARTA.
//use WASD to control sebot moving
void serialRXA(serial_t *s, char data) {
    numRXA ++;
    if (data == 'a') {                              // turn left
        turnrate = turnrate - 0.2;
    } else if (data == 'd') {                       // turn right
        turnrate = turnrate + 0.2;
    } else if (data == 'w') {                       // go forward
        FwdBackOffset = FwdBackOffset - 0.2;
    } else if (data == 's') {                       // go backward
        FwdBackOffset = FwdBackOffset + 0.2;
    } else if (data == 'r') {                       // reset position
        turnrate = 0;
        FwdBackOffset = 0;
        reset = 1;
    } else if (data == 't') {                       // start auto moving
        stage = 1;
    } else {                                      // reset speed
        turnrate = 0;
        FwdBackOffset = 0;
    }
}


void setupSpib(void) //Call this function in main() somewhere after the DINT; line of code.
{
    int16_t temp = 0;
```

```
    int16_t i_temp = 0;
    //Step 1.
    // cut and paste here all the SpibRegs initializations you found for part 3. Change so that 16 bits are
    //transmitted each TX FIFO write and change the delay in between each transfer to 0.
    //----------------------------------------------------------------------------------------------------------


    GPIO_SetupPinMux(2, GPIO_MUX_CPU1, 0); // Set as GPIO2 and used as DAN777 SS
    GPIO_SetupPinOptions(2, GPIO_OUTPUT, GPIO_PUSHPULL); // Make GPIO2 an Output Pin
    GpioDataRegs.GPASET.bit.GPIO2 = 1; //Initially Set GPIO2/SS High so DAN777 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 prequalifier for SPI PINS
    GpioCtrlRegs.GPCQSEL1.bit.GPIO64 = 3; // The prequalifier eliminates short noise spikes
    GpioCtrlRegs.GPCQSEL1.bit.GPIO65 = 3; // by making sure the serial pin stays low for 3 clock periods.
    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 DAN777 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 SCLK'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 = 0x00; //Set delay between transmits to 16 spi clocks. Needed by DAN777 chip
        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 don't think this is needed. Need to Test
        SpibRegs.SPIFFRX.bit.RXFFIL =2; //Interrupt Level to 2 words or more received into FIFO causes interrupt
```

```


        //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

        // To address 00x13 write 0x00
        SpibRegs.SPITXBUF = 0x1300;
        // To address 00x14 write 0x00
        // To address 00x15 write 0x00
        SpibRegs.SPITXBUF = 0x0;
        // To address 00x16 write 0x00
        // To address 00x17 write 0x00
        SpibRegs.SPITXBUF = 0x0;
        // To address 00x18 write 0x00
        // To address 00x19 write 0x13
        SpibRegs.SPITXBUF = 0x0013;
        // To address 00x1A write 0x02
        // To address 00x1B write 0x00
        SpibRegs.SPITXBUF = 0x0200;
        // To address 00x1C write 0x08
        // To address 00x1D write 0x06
        SpibRegs.SPITXBUF = 0x0806;
        // To address 00x1E write 0x00
        // To address 00x1F write 0x00
        SpibRegs.SPITXBUF = 0x0;

        // 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
        for(i_temp=0; i_temp<7; i_temp++){
            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 13 registers
            // To address 00x23 write 0x00
            SpibRegs.SPITXBUF = 0x2300;
            // To address 00x24 write 0x40
            // To address 00x25 write 0x8C
            SpibRegs.SPITXBUF = 0x408C;
            // To address 00x26 write 0x02
            // To address 00x27 write 0x88
            SpibRegs.SPITXBUF = 0x0288;
            // To address 00x28 write 0x0C
            // To address 00x29 write 0x0A
            SpibRegs.SPITXBUF = 0x0C0A;


            // 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 = 0x2A81;
            // 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;
        SpibRegs.SPITXBUF = (0x7700 | 0x00EB); // 0x7700
        while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
        GpioDataRegs.GPCSET.bit.GPIO66 = 1;
        temp = SpibRegs.SPIRXBUF;
        DELAY_US(10);

        GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
        SpibRegs.SPITXBUF = (0x7800 | 0x0012); // 0x7800
        while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
        GpioDataRegs.GPCSET.bit.GPIO66 = 1;
        temp = SpibRegs.SPIRXBUF;
        DELAY_US(10);

        GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
        SpibRegs.SPITXBUF = (0x7A00 | 0x0010); // 0x7A00
        while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
        GpioDataRegs.GPCSET.bit.GPIO66 = 1;
        temp = SpibRegs.SPIRXBUF;
        DELAY_US(10);

        GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
        SpibRegs.SPITXBUF = (0x7B00 | 0x00FA); // 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 | 0x0021); // 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 | 0x0050); // 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;
    }



    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.QEPCTL.bit.QPEN = 1; // Enable EQep
        EQep1Regs.QPOSCNT = 0;
        // 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.QEPCTL.bit.QPEN = 1; // Enable EQep
        EQep2Regs.QPOSCNT = 0;
    }

    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
        // 20 North South magnet poles in the encoder disk so 20 square waves per one revolution of the
        // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so 80 counts per one rev
        // of the DC motor's back shaft. Then the gear motor's gear ratio is 18.7.
        return (raw*(2*PI/80.0)/18.7);
    }

    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
        // 20 North South magnet poles in the encoder disk so 20 square waves per one revolution of the
        // DC motor's back shaft. Then Quadrature Decoder mode multiplies this by 4 so 80 counts per one rev
        // of the DC motor's back shaft. Then the gear motor's gear ratio is 18.7.
        return (raw*(2*PI/80.0)/18.7);
    }

    void setEPWM6A (float controleffort){

        if (controleffort >= 10){
            controleffort = 10;
        }
        else if (controleffort <= -10){
            controleffort = -10;
        }

```

```
        if (controleffort >= 0){
            GpioDataRegs.GPASET.bit.GPIO29 = 1;
        }
        else{
            GpioDataRegs.GPACLEAR.bit.GPIO29 = 1;
        }
        EPwm6Regs.CMPA.bit.CMPA = 2500 * (fabs(controleffort) /
10.0);
    }

    void setEPWM6B (float controleffort){

        if (controleffort >= 10){
            controleffort = 10;

        }
        else if (controleffort <= -10){
            controleffort = -10;
        }

        if (controleffort >= 0){
            GpioDataRegs.GPBSET.bit.GPIO32 = 1;
        }
        else{
            GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;
        }
        EPwm6Regs.CMPB.bit.CMPB = 2500 * (fabs(controleffort) /
10.0);
    }
```