

# 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.

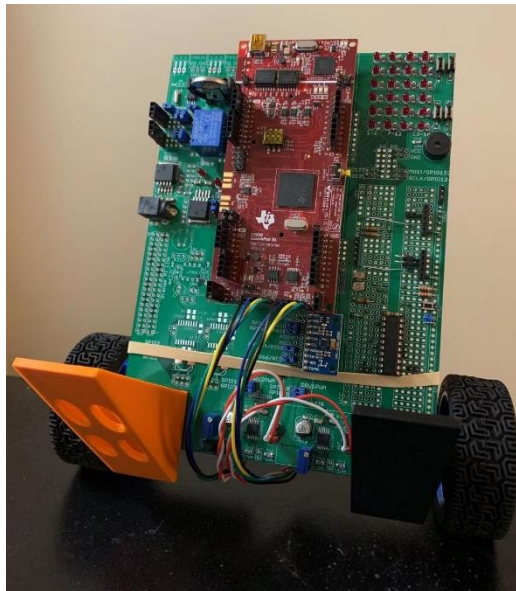


Figure 1 self-balanced Segbot

## 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.  $x_m$  and  $y_m$  are the position of the Segbot on the X-Y plane which we would like to know.  $R_w$  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.  $x_T$  and  $y_T$  are the target position, 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.

```
1. // (Final Project) position tracking variables
2. int stage = 0; // moving stage
3. float xm = 0.0; // m, Wheel axle midpoint position in X axis
4. float ym = 0.0; // m, Wheel axle midpoint position in Y axis
5. float xT = 0.0; // m, target position in X axis
6. float yT = 0.0; // m, target position in Y axis
7. float xm_dot = 0.0; // m/s, Midpoint velocity on X axis
8. float ym_dot = 0.0; // m/s, Midpoint velocity on Y axis
9. float xm_dot_1 = 0.0;
10. float ym_dot_1 = 0.0;
11. 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)
12. float theta_left = 0.0; // Rad, Left wheel angle (make it possible
    to reset LeftWheel and RightWheel)
13. float theta_right = 0.0; // Rad, Right wheel angle
14. float theta_dot = 0.0; // Rad/s, Average turning speed of wheels
15. float theta = 0.0; // Rad, Average angle of left and right wheel
16. float theta_1 = 0.0;
17. float phi = 0.0; // Rad, Body yaw angle
18. float phi_T = 0.0; // Rad, direction from position now to target
19. float W = 0.182; // m, Distance between the centers of the two
    wheels
```

```
20. 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.

```
1. //EPWM5
2. EALLOW;
3. EPwm5Regs.ETSEL.bit.SOCAEN = 0; // Disable SOC on A group
4. EPwm5Regs.TBCTL.bit.CTRMODE = 3; // freeze counter
5. EPwm5Regs.ETSEL.bit.SOCASEL = 2; // Select Event when counter equal
    to PRD
6. EPwm5Regs.ETPS.bit.SOCAPRD = 1; // Generate pulse on 1st event
    ("pulse" is the same as "trigger")
7. EPwm5Regs.TBCTR = 0x0; // Clear counter
8. EPwm5Regs.TBPHS.bit.TBPHS = 0x0000; // Phase is 0
9. EPwm5Regs.TBCTL.bit.PHSEN = 0; // Disable phase loading
10. EPwm5Regs.TBCTL.bit.CLKDIV = 0; // divide by 1 50Mhz Clock
11. //ME461, Lab 3 Lab 5, Page 3 of 11
12. EPwm5Regs.TBPRD = 50000; // Set Period to 1ms sample. Input clock
    is 50MHz.
13. // Notice here that we are not setting CMPA or CMPB because we are
    not using the PWM signal
14. EPwm5Regs.ETSEL.bit.SOCAEN = 1; //enable SOCA
15. EPwm5Regs.TBCTL.bit.CTRMODE = 0; //unfreeze, and enter up count
    mode
16. 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.

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

```

9.     AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT,
ADC_SIGNALMODE_SINGLE); //read calibration settings
10.    //Set pulse positions to late
11.    AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
12.    AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
13.    AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
14.    AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;
15.    //power up the ADCs
16.    AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
17.    AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
18.    AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
19.    AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;
20.    //delay for 1ms to allow ADC time to power up
21.    DELAY_US(1000);
22.    //Select the channels to convert and end of conversion flag
23.    //Many statements commented out, To be used when using ADCA or
ADCB
24.    //ME461, Lab 3 Lab 5, Page 4 of 11
25.
26.    //ADCA
27.    AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2; //SOC0 will convert Channel
you choose Does not have to be A0
28.    AdcaRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1
SYSCLK cycles = 75ns
29.    AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0xD; // EPWM5 ADCSOCA or
another trigger you choose will trigger SOC0
30.
31.    AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3; //SOC1 will convert Channel
you choose Does not have to be A1
32.    AdcaRegs.ADCSOC1CTL.bit.ACQPS = 14; //sample window is acqps + 1
SYSCLK cycles = 75ns
33.    AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0xD; // EPWM5 ADCSOCA or
another trigger you choose will trigger SOC1
34.
35.    AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //set to last SOC that is
converted and it will set INT1 flag ADCA1
36.    AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1; //enable INT1 flag
37.    AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
cleared
38.    EDIS;
39.
40.    EALLOW;
41.    DacARegs.DACOUTEN.bit.DACOUTEN = 1; //enable dacA output-->uses
ADCINA0

```

```

42.     DacARegs.DACCTL.bit.LOADMODE = 0; //load on next sysclk
43.     DacARegs.DACCTL.bit.DACREFSEL = 1; //use ADC VREF as reference
        voltage
44.     DacBRegs.DACOUTEN.bit.DACOUTEN = 1; //enable dacB output-->uses
        ADCINA1
45.     DacBRegs.DACCTL.bit.LOADMODE = 0; //load on next sysclk
46.     DacBRegs.DACCTL.bit.DACREFSEL = 1; //use ADC VREF as reference
        voltage
47.     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.

```

1.     //initialize the PWM register 6A, 6B
2.     EPwm6Regs.TBCTL.bit.CTRMODE = 0;
3.     EPwm6Regs.TBCTL.bit.FREE_SOFT = 2;
4.     EPwm6Regs.TBCTL.bit.PHSEN = 0;
5.     EPwm6Regs.TBCTL.bit.CLKDIV = 0;
6.
7.     EPwm6Regs.TBCTR = 0;
8.     EPwm6Regs.TBPRD = 2500;
9.     EPwm6Regs.TBPHS.bit.TBPHS = 0;
10.    EPwm6Regs.CMPA.bit.CMPA = 0;
11.    EPwm6Regs.AQCTLA.bit.CAU = 1;
12.    EPwm6Regs.AQCTLA.bit.ZRO = 2;
13.
14.    EPwm6Regs.CMPB.bit.CMPB = 0;
15.    EPwm6Regs.AQCTLB.bit.CBU = 1;
16.    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.  $U_{bal} = -K_1 \cdot \text{tilt} - K_2 \cdot \text{gyrorate} - K_3 \cdot (\text{velLeft} + \text{velRight})/2$  is the control law we used here.  $U_{bal}$  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).

```

1. // SWI_isr, Using this interrupt as a Software started interrupt
2. __interrupt void SWI_isr(void) {
3.
4.     // These three lines of code allow SWI_isr, to be interrupted by
   other interrupt functions
5.     // making it lower priority than all other Hardware interrupts.
6.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
7.     asm("        NOP");           // Wait one cycle
8.     EINT;                         // Clear INTM to enable
   interrupts
9.
10.    // Insert SWI ISR Code here.....
11.
12.    turnref = turnref_1 + 0.002 * (turnrate + turnrate_1);
13.
14.    WhlDiff = LeftWheel - RightWheel;
15.    vel_WhlDiff = 0.333 * vel_WhlDiff_1 + 2.0/(3.0*0.004)*(WhlDiff -
   WhlDiff_1);
16.    eDiff = turnref - WhlDiff;
17.    intDiff = intDiff_1 + 0.002 * (eDiff + eDiff_1);
18.
19.    turn = Kp*eDiff + Ki*intDiff - Kd*vel_WhlDiff;
20.
21.    if(fabs(turn)>3){
22.        intDiff = intDiff_1;
23.    }
24.    if(turn > 4.0){
25.        turn = 4.0;
26.    }
27.    if(turn < -4.0){
28.        turn = -4.0;
29.    }
30.
31.    WhlDiff_1 = WhlDiff;
32.    vel_WhlDiff_1 = vel_WhlDiff;
33.    eDiff_1 = eDiff;
34.    intDiff_1 = intDiff;
35.    turnref_1 = turnref;
36.    turnrate_1 = turnrate;
37.
38.    v_l = 0.6 * v_l_1 + 100 * (LeftWheel - LeftWheel_1);
39.    v_r = 0.6 * v_r_1 + 100 * (RightWheel - RightWheel_1);
40.

```

```

41.     Ubal = -K1 * tilt_value - K2 * gyro_value - K3 * (v_l +
      v_r)/2.0;
42.
43.     v_l_1 = v_l;
44.     v_r_1 = v_r;
45.
46.     uLeft = (Ubal/2.0) + turn + FwdBackOffset;
47.     uRight = (Ubal/2.0) - turn + FwdBackOffset;
48.
49.     setEPWM6A(uLeft);
50.     setEPWM6B(-uRight);
51.
52.     // Final Project add code here
53.     if (reset == 1) {
54.         xm = 0.0;
55.         ym = 0.0;
56.         xm_dot = 0.0;
57.         ym_dot = 0.0;
58.         xm_dot_1 = 0.0;
59.         ym_dot_1 = 0.0;
60.         theta_dot = 0.0;
61.         theta = 0.0;
62.         theta_1 = 0.0;
63.         phi = 0.0;
64.         theta_left = 0.0;
65.         theta_right = 0.0;
66.         reset = 0;
67.     }
68.     theta_left = theta_left + (LeftWheel - LeftWheel_1);
69.     theta_right = theta_right + (RightWheel - RightWheel_1);
70.     theta = 0.5 * (theta_left + theta_right);
71.     phi = Rw / W * (theta_right - theta_left);
72.     theta_dot = (theta - theta_1) / 0.004;
73.     xm_dot = Rw * theta_dot * cos(phi);
74.     ym_dot = Rw * theta_dot * sin(phi);
75.     xm = xm + 0.004 * (xm_dot + xm_dot_1);
76.     ym = ym + 0.004 * (ym_dot + ym_dot_1);
77.
78.     theta_1 = theta;
79.     xm_dot = xm_dot_1;
80.     ym_dot = ym_dot_1;
81.     LeftWheel_1 = LeftWheel;
82.     RightWheel_1 = RightWheel;
83.

```

```

84.     // Final project auto move part
85.     if (stage == 0) {    // prepare stage
86.         phi_T = atan2(yT-ym,xT-xm); // specify the direction towards
            the target
87.     }
88.     if (stage == 1) {    // turning stage
89.         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
90.             turnrate = 0.0;
91.             stage = 2;
92.         }
93.         else if (phi_T > 0) {
94.             turnrate = -1.0; // turn left to face the target
95.         }
96.         else if (phi_T < 0) {
97.             turnrate = 1.0; // turn right to face the target
98.         }
99.     }
100.    if (stage == 2) {    // moving stage
101.        if ((fabs(xm-xT) < 0.01) && (fabs(ym-yT) < 0.01)) {
102.            turnrate = 0.0;
103.            FwdBackOffset = 0.0;
104.            stage = 3;
105.        }
106.        else {
107.            FwdBackOffset = -1.5;
108.        }
109.    }
110.    numSWIcalls++;
111.    DINT;
112. }

```

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.

```

1. __interrupt void SPIB_isr(void){
2.
3.     int16_t temp1 = 0;
4.     numSPICalls++;
5.

```



```

6.     temp1 = SpibRegs.SPIRXBUF;
7.     x_accel = SpibRegs.SPIRXBUF;
8.     y_accel = SpibRegs.SPIRXBUF;
9.     z_accel = SpibRegs.SPIRXBUF;
10.    temp1 = SpibRegs.SPIRXBUF;
11.    x_gyro = SpibRegs.SPIRXBUF;
12.    y_gyro = SpibRegs.SPIRXBUF;
13.    z_gyro = SpibRegs.SPIRXBUF;
14.    //
15.    GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Set GPIO 66 to end Slave
    Select of MPU.
16.
17.    gyrox = x_gyro * 250.0 / 32767.0;
18.    gyroy = y_gyro * 250.0 / 32767.0;
19.    gyroz = y_gyro * 250.0 / 32767.0;
20.    accelx = x_accel * 4.0 / 32767.0;
21.    accely = y_accel * 4.0 / 32767.0;
22.    accelz = z_accel * 4.0 / 32767.0;
23.
24.    //Code to be copied into SPIB_ISR interrupt function after the
    IMU measurements have been collected.
25.    if(calibration_state == 0){
26.        calibration_count++;
27.        if (calibration_count == 2000) {
28.            calibration_state = 1;
29.            calibration_count = 0;
30.        }
31.    }
32.
33.    else if(calibration_state == 1){
34.        accelx_offset+=accelx;
35.        accely_offset+=accely;
36.        accelz_offset+=accelz;
37.        gyrox_offset+=gyrox;
38.        gyroy_offset+=gyroy;
39.        gyroz_offset+=gyroz;
40.        calibration_count++;
41.        if (calibration_count == 2000) {
42.            calibration_state = 2;
43.            accelx_offset/=2000.0;
44.            accely_offset/=2000.0;
45.            accelz_offset/=2000.0;
46.            gyrox_offset/=2000.0;
47.            gyroy_offset/=2000.0;

```

```

48.         gyro_z_offset/=2000.0;
49.         calibration_count = 0;
50.         doneCal = 1;
51.     }
52. }
53.
54. else if(calibration_state == 2){
55.     accelx -=(accelx_offset);
56.     accely -=(accely_offset);
57.     accelz -=(accelz_offset-accelzBalancePoint);
58.     gyro_x -= gyro_x_offset;
59.     gyro_y -= gyro_y_offset;
60.     gyro_z -= gyro_z_offset;
61.
62.     /*-----Kalman Filtering code start-----
-----*/
63.     float tiltrate = (gyro_x*PI)/180.0; // rad/s
64.     float pred_tilt, z, y, S;
65.
66.     tilt_rate = tiltrate;
67.
68.     // Prediction Step
69.     pred_tilt = kalman_tilt + T*tiltrate;
70.     pred_P = kalman_P + Q;
71.
72.     // Update Step
73.     z = -accelz;
74.     y = z - pred_tilt;
75.     S = pred_P + R;
76.     kalman_K = pred_P/S;
77.     kalman_tilt = pred_tilt + kalman_K*y;
78.     kalman_P = (1 - kalman_K)*pred_P;
79.     SpibNumCalls++;
80.
81.     // Kalman Filter used
82.     tilt_array[SpibNumCalls] = kalman_tilt;
83.     gyro_array[SpibNumCalls] = tiltrate;
84.     LeftWheelArray[SpibNumCalls] = readEncLeft();
85.     RightWheelArray[SpibNumCalls] = -readEncRight();
86.
87.     if (SpibNumCalls >= 3) { // should never be greater than 3
88.         tilt_value = (tilt_array[0] + tilt_array[1] +
            tilt_array[2] + tilt_array[3])/4.0;

```

```

89.         gyro_value = (gyro_array[0] + gyro_array[1] +
        gyro_array[2] + gyro_array[3])/4.0;
90.         LeftWheel=(LeftWheelArray[0]+LeftWheelArray[1]+LeftWheelArray[2]+Left
        WheelArray[3])/4.0;
91.         RightWheel=(RightWheelArray[0]+RightWheelArray[1]+RightWheelArray[2]+
        RightWheelArray[3])/4.0;
92.         SpibNumCalls = -1;
93.         PieCtrlRegs.PIEIFR12.bit.INTx9 = 1; // Manually cause the
        interrupt for the SWI
94.     }
95. }
96.
97.     timecount++;
98.     if((timecount%200) == 0)
99.     {
100.         if(doneCal == 0) {
101.             GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1; // Blink Blue LED
            while calibrating
102.         }
103.         GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Always Block Red
            LED
104.         UARTPrint = 1; // Tell While loop to print
105.     }
106.
107.     SpibRegs.SPIFFRX.bit.RXFFOVFCLR=1; // Clear Overflow flag
108.     SpibRegs.SPIFFRX.bit.RXFFINTCLR=1; // Clear Interrupt flag
109.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;
110. }

```

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.

```

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

```

```

9.      } else if (data == 'w') {                                // go forward
10.          FwdBackOffset = FwdBackOffset - 0.2;
11.      } else if (data == 's') {                                // go backward
12.          FwdBackOffset = FwdBackOffset + 0.2;
13.      } else if (data == 'r') {                                // reset position
14.          turnrate = 0;
15.          FwdBackOffset = 0;
16.          reset = 1;
17.      } else {                                                // reset speed
18.          turnrate = 0;
19.          FwdBackOffset = 0;
20.      }
21.  }

```

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  $\text{raw\_value} * (2 * \text{PI} / 80.0) / 18.7$ .

```

1. float readEncLeft(void) {
2.     int32_t raw = 0;
3.     uint32_t QEP_maxvalue = 0xFFFFFFFFU; //4294967295U
4.     raw = EQep1Regs.QPOSCNT;
5.     if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue; // I don't think
        this is needed and never true
6.     // 20 North South magnet poles in the encoder disk so 20 square
        waves per one revolution of the
7.     // DC motor's back shaft. Then Quadrature Decoder mode multiplies
        this by 4 so 80 counts per one rev
8.     // of the DC motor's back shaft. Then the gear motor's gear ratio
        is 18.7.
9.     return (raw*(2*PI/80.0)/18.7);
10. }
11.
12. float readEncRight(void) {
13.     int32_t raw = 0;
14.     uint32_t QEP_maxvalue = 0xFFFFFFFFU; //4294967295U -1 32bit
        signed int
15.     raw = EQep2Regs.QPOSCNT;
16.     if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue; // I don't think
        this is needed and never true

```

```

17.      // 20 North South magnet poles in the encoder disk so 20 square
        waves per one revolution of the
18.      // DC motor's back shaft. Then Quadrature Decoder mode
        multiplies this by 4 so 80 counts per one rev
19.      // of the DC motor's back shaft. Then the gear motor's gear
        ratio is 18.7.
20.      return (raw*(2*PI/80.0)/18.7);
21.  }

```

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.

```

1. void setEPWM6A (float controleffort){
2.
3.     if (conroleffort >= 10){
4.         controleffort = 10;
5.     }
6.     else if (conroleffort <= -10){
7.         controleffort = -10;
8.     }
9.
10.    if (conroleffort >= 0){
11.        GpioDataRegs.GPASET.bit.GPIO29 = 1;
12.    }
13.    else{
14.        GpioDataRegs.GPACLEAR.bit.GPIO29 = 1;
15.    }
16.    EPwm6Regs.CMPA.bit.CMPA = 2500 * (fabs(conroleffort) / 10.0);
17. }
18.
19. void setEPWM6B (float controleffort){
20.
21.     if (conroleffort >= 10){
22.         controleffort = 10;
23.     }
24.     }
25.     else if (conroleffort <= -10){
26.         controleffort = -10;
27.     }
28.
29.     if (conroleffort >= 0){
30.         GpioDataRegs.GPBSET.bit.GPIO32 = 1;

```

```

31.     }
32.     else{
33.         GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;
34.     }
35.     EPwm6Regs.CMPB.bit.CMPB = 2500 * (fabs(controleffort) / 10.0);
36. }

```

#### 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:

```

1. // Included Files
2. #include <stdio.h>
3. #include <stdlib.h>
4. #include <stdarg.h>
5. #include <string.h>
6. #include <math.h>
7. #include <limits.h>

```

```
8. #include "F28x_Project.h"
9. #include "driverlib.h"
10. #include "device.h"
11. #include "f28379dSerial.h"
12.
13. #define PI          3.1415926535897932384626433832795
14. #define TWOPI       6.283185307179586476925286766559
15. #define HALFPI      1.5707963267948966192313216916398
16. #define g           9.8
17.
18. // Interrupt Service Routines predefinition
19. __interrupt void cpu_timer0_isr(void);
20. __interrupt void cpu_timer1_isr(void);
21. __interrupt void cpu_timer2_isr(void);
22. __interrupt void SWI_isr(void);
23. __interrupt void ADCA_ISR (void);
24. __interrupt void SPIB_isr(void);
25.
26. void serialRXA(serial_t *s, char data);
27. void setupSpib(void);
28. void init_eQEPs(void);
29. float readEncLeft(void);
30. float readEncRight(void);
31. void setEPWM6A (float);
32. void setEPWM6B (float);
33.
34. // Count variables
35. uint32_t numTimer0calls = 0;
36. uint32_t numSWIcalls = 0;
37. uint32_t numRXA = 0;
38. uint16_t UARTPrint = 0;
39. uint32_t numADCAcalls = 0;
40. uint32_t numSPIcalls = 0;
41.
42. float out1 = 0.0;
43. float out2 = 0.0;
44. float analogvalue0 = 0.0;
45. float analogvalue1 = 0.0;
46. float adcinput1;
47. float adcinput2;
48. float LeftWheel = 0.0;
49. float RightWheel = 0.0;
50. float LeftWheel_1 = 0.0;
51. float RightWheel_1 = 0.0;
```

```
52. float LW_factor = 9.8;
53. float RW_factor = 9.8;
54. float uLeft = 0.0;
55. float uRight = 0.0;
56. float XLeft_K = 0.0;
57. float XRight_K = 0.0;
58. float XLeft_K_1 = 0.0;
59. float XRight_K_1 = 0.0;
60. float VLeft_K = 0.0;
61. float VRight_K = 0.0;
62. float e_K_L = 0.0;
63. float e_K_R = 0.0;
64. float e_K_1_L = 0.0;
65. float e_K_1_R = 0.0;
66. float I_K_L = 0.0;
67. float I_K_R = 0.0;
68. float I_K_1_L = 0.0;
69. float I_K_1_R = 0.0;
70. float Vref = 1;
71. float Kpt = 3.0;
72. float e_turn = 0.0;
73.
74. int32_t digital0 = 0;
75. int32_t digital1 = 0;
76.
77. int16_t x_gyro = 0;
78. int16_t y_gyro = 0;
79. int16_t z_gyro = 0;
80. int16_t x_accel = 0;
81. int16_t y_accel = 0;
82. int16_t z_accel = 0;
83.
84. float gyro_x = 0.0;
85. float gyro_y = 0.0;
86. float gyro_z = 0.0;
87. float accel_x = 0.0;
88. float accel_y = 0.0;
89. float accel_z = 0.0;
90.
91. // Needed global Variables
92. float accel_x_offset = 0;
93. float accel_y_offset = 0;
94. float accel_z_offset = 0;
95. float gyro_x_offset = 0;
```



```
96. float gyroy_offset = 0;
97. float gyroz_offset = 0;
98. float accelzBalancePoint = -.762;
99. int16 IMU_data[9];
100. int16_t doneCal = 0;
101. float tilt_value = 0;
102. float tilt_array[4] = {0, 0, 0, 0};
103. float gyro_value = 0;
104. float gyro_array[4] = {0, 0, 0, 0};
105. float LeftWheelArray[4] = {0,0,0,0};
106. float RightWheelArray[4] = {0,0,0,0};
107.
108. //balance control
109. float Ubal = 0.0;
110. float tilt_rate = 0.0;
111. float v_l = 0.0;
112. float v_r = 0.0;
113. float v_l_1 = 0.0;
114. float v_r_1 = 0.0;
115.
116. float K1 = -30.0;
117. float K2 = -2.8;
118. float K3 = -1.0;
119.
120. //turn control
121. float WhlDiff = 0.0;
122. float WhlDiff_1 = 0.0;
123. float vel_WhlDiff = 0.0;
124. float vel_WhlDiff_1 = 0.0;
125. float intDiff = 0.0;
126. float intDiff_1 = 0.0;
127. float eDiff = 0.0;
128. float eDiff_1 = 0.0;
129.
130. float turnref = 0.0;
131. float turnref_1 = 0.0;
132. float turn = 0.0;
133. float turnrate = 0.0;
134. float turnrate_1 = 0.0;
135. float FwdBackOffset = 0.0;
136.
137. float Kp = 3.0;
138. float Ki = 20.0;
139. float Kd = 0.08;
```

```

140.
141. // Kalman Filter
142. float T = 0.001; //sample rate, 1ms
143. float Q = 0.01; // made global to enable changing in runtime
144. float R = 25000; //50000;
145. float kalman_tilt = 0;
146. float kalman_P = 22.365;
147. int16_t SpibNumCalls = -1;
148. float pred_P = 0;
149. float kalman_K = 0;
150. int32_t timecount = 0;
151. int16_t calibration_state = 0;
152. int32_t calibration_count = 0;
153.
154. // (Final Project) position tracking variables
155. int stage = 0; // moving stage
156. float xm = 0.0; // m, Wheel axle midpoint position in X axis
157. float ym = 0.0; // m, Wheel axle midpoint position in Y axis
158. float xT = 0.0; // m, target position in X axis
159. float yT = 0.0; // m, target position in Y axis
160. float xm_dot = 0.0; // m/s, Midpoint velocity on X axis
161. float ym_dot = 0.0; // m/s, Midpoint velocity on Y axis
162. float xm_dot_1 = 0.0;
163. float ym_dot_1 = 0.0;
164. 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)
165. float theta_left = 0.0; // Rad, Left wheel angle (make it possible
    to reset LeftWheel and RightWheel)
166. float theta_right = 0.0; // Rad, Right wheel angle
167. float theta_dot = 0.0; // Rad/s, Average turning speed of wheels
168. float theta = 0.0; // Rad, Average angle of left and right wheel
169. float theta_1 = 0.0;
170. float phi = 0.0; // Rad, Body yaw angle
171. float phi_T = 0.0; // Rad, direction from position now to target
172. float W = 0.182; // m, Distance between the centers of the two
    wheels
173. int reset = 0; // Reset flag for reset position to zero (also set
    LeftWheel and RightWheel to zero)
174.
175. void setDACA(float dacouta0) {
176.     if (dacouta0 > 3.0) dacouta0 = 3.0;
177.     if (dacouta0 < 0.0) dacouta0 = 0.0;

```

```

178.     DacbRegs.DACVALS.bit.DACVALS = 4095.0 / 3.0 * dacouta0; //
        perform scaling of 0-3 to 0-4095
179. }
180.
181. void setDACB(float dacouta1) {
182.     if (dacouta1 > 3.0) dacouta1 = 3.0;
183.     if (dacouta1 < 0.0) dacouta1 = 0.0;
184.     DacbRegs.DACVALS.bit.DACVALS = 4095.0 / 3.0 * dacouta1; //
        perform scaling of 0-3 to 0-4095
185. }
186.
187. void main(void){
188.     // PLL, WatchDog, enable Peripheral Clocks
189.     // This example function is found in the F2837xD_SysCtrl.c file.
190.     InitSysCtrl();
191.
192.     InitGpio();
193.
194.     // Blue LED on LuanchPad
195.     GPIO_SetupPinMux(31, GPIO_MUX_CPU1, 0);
196.     GPIO_SetupPinOptions(31, GPIO_OUTPUT, GPIO_PUSHPULL);
197.     GpioDataRegs.GPASET.bit.GPIO31 = 1;
198.
199.     // Red LED on LaunchPad
200.     GPIO_SetupPinMux(34, GPIO_MUX_CPU1, 0);
201.     GPIO_SetupPinOptions(34, GPIO_OUTPUT, GPIO_PUSHPULL);
202.     GpioDataRegs.GPBSET.bit.GPIO34 = 1;
203.
204.     // LED1 and PWM Pin
205.     GPIO_SetupPinMux(22, GPIO_MUX_CPU1, 0);
206.     GPIO_SetupPinOptions(22, GPIO_OUTPUT, GPIO_PUSHPULL);
207.     GpioDataRegs.GPACLEAR.bit.GPIO22 = 1;
208.
209.     // LED2
210.     GPIO_SetupPinMux(52, GPIO_MUX_CPU1, 0);
211.     GPIO_SetupPinOptions(52, GPIO_OUTPUT, GPIO_PUSHPULL);
212.     GpioDataRegs.GPBCLEAR.bit.GPIO52 = 1;
213.
214.     // LED3
215.     GPIO_SetupPinMux(67, GPIO_MUX_CPU1, 0);
216.     GPIO_SetupPinOptions(67, GPIO_OUTPUT, GPIO_PUSHPULL);
217.     GpioDataRegs.GPCCLEAR.bit.GPIO67 = 1;
218.
219.     // LED4

```

```
220.     GPIO_SetupPinMux(94, GPIO_MUX_CPU1, 0);
221.     GPIO_SetupPinOptions(94, GPIO_OUTPUT, GPIO_PUSHPULL);
222.     GpioDataRegs.GPCCLEAR.bit.GPIO94 = 1;
223.
224.     // LED5
225.     GPIO_SetupPinMux(95, GPIO_MUX_CPU1, 0);
226.     GPIO_SetupPinOptions(95, GPIO_OUTPUT, GPIO_PUSHPULL);
227.     GpioDataRegs.GPCCLEAR.bit.GPIO95 = 1;
228.
229.     // LED6
230.     GPIO_SetupPinMux(97, GPIO_MUX_CPU1, 0);
231.     GPIO_SetupPinOptions(97, GPIO_OUTPUT, GPIO_PUSHPULL);
232.     GpioDataRegs.GPDCLEAR.bit.GPIO97 = 1;
233.
234.     // LED7
235.     GPIO_SetupPinMux(111, GPIO_MUX_CPU1, 0);
236.     GPIO_SetupPinOptions(111, GPIO_OUTPUT, GPIO_PUSHPULL);
237.     GpioDataRegs.GPDCLEAR.bit.GPIO111 = 1;
238.
239.     // LED8
240.     GPIO_SetupPinMux(130, GPIO_MUX_CPU1, 0);
241.     GPIO_SetupPinOptions(130, GPIO_OUTPUT, GPIO_PUSHPULL);
242.     GpioDataRegs.GPECLEAR.bit.GPIO130 = 1;
243.
244.     // LED9
245.     GPIO_SetupPinMux(131, GPIO_MUX_CPU1, 0);
246.     GPIO_SetupPinOptions(131, GPIO_OUTPUT, GPIO_PUSHPULL);
247.     GpioDataRegs.GPECLEAR.bit.GPIO131 = 1;
248.
249.     // LED10
250.     GPIO_SetupPinMux(4, GPIO_MUX_CPU1, 0);
251.     GPIO_SetupPinOptions(4, GPIO_OUTPUT, GPIO_PUSHPULL);
252.     GpioDataRegs.GPACLEAR.bit.GPIO4 = 1;
253.
254.     // LED11
255.     GPIO_SetupPinMux(5, GPIO_MUX_CPU1, 0);
256.     GPIO_SetupPinOptions(5, GPIO_OUTPUT, GPIO_PUSHPULL);
257.     GpioDataRegs.GPACLEAR.bit.GPIO5 = 1;
258.
259.     // LED12
260.     GPIO_SetupPinMux(6, GPIO_MUX_CPU1, 0);
261.     GPIO_SetupPinOptions(6, GPIO_OUTPUT, GPIO_PUSHPULL);
262.     GpioDataRegs.GPACLEAR.bit.GPIO6 = 1;
263.
```

```
264. // LED13
265. GPIO_SetupPinMux(7, GPIO_MUX_CPU1, 0);
266. GPIO_SetupPinOptions(7, GPIO_OUTPUT, GPIO_PUSHPULL);
267. GpioDataRegs.GPACLEAR.bit.GPIO7 = 1;
268.
269. // LED14
270. GPIO_SetupPinMux(8, GPIO_MUX_CPU1, 0);
271. GPIO_SetupPinOptions(8, GPIO_OUTPUT, GPIO_PUSHPULL);
272. GpioDataRegs.GPACLEAR.bit.GPIO8 = 1;
273.
274. // LED15
275. GPIO_SetupPinMux(9, GPIO_MUX_CPU1, 0);
276. GPIO_SetupPinOptions(9, GPIO_OUTPUT, GPIO_PUSHPULL);
277. GpioDataRegs.GPACLEAR.bit.GPIO9 = 1;
278.
279. // LED16
280. GPIO_SetupPinMux(24, GPIO_MUX_CPU1, 0);
281. GPIO_SetupPinOptions(24, GPIO_OUTPUT, GPIO_PUSHPULL);
282. GpioDataRegs.GPACLEAR.bit.GPIO24 = 1;
283.
284. // LED17
285. GPIO_SetupPinMux(25, GPIO_MUX_CPU1, 0);
286. GPIO_SetupPinOptions(25, GPIO_OUTPUT, GPIO_PUSHPULL);
287. GpioDataRegs.GPACLEAR.bit.GPIO25 = 1;
288.
289. // LED18
290. GPIO_SetupPinMux(26, GPIO_MUX_CPU1, 0);
291. GPIO_SetupPinOptions(26, GPIO_OUTPUT, GPIO_PUSHPULL);
292. GpioDataRegs.GPACLEAR.bit.GPIO26 = 1;
293.
294. // LED19
295. GPIO_SetupPinMux(27, GPIO_MUX_CPU1, 0);
296. GPIO_SetupPinOptions(27, GPIO_OUTPUT, GPIO_PUSHPULL);
297. GpioDataRegs.GPACLEAR.bit.GPIO27 = 1;
298.
299. // LED20
300. GPIO_SetupPinMux(60, GPIO_MUX_CPU1, 0);
301. GPIO_SetupPinOptions(60, GPIO_OUTPUT, GPIO_PUSHPULL);
302. GpioDataRegs.GPBCLEAR.bit.GPIO60 = 1;
303.
304. // LED21
305. GPIO_SetupPinMux(61, GPIO_MUX_CPU1, 0);
306. GPIO_SetupPinOptions(61, GPIO_OUTPUT, GPIO_PUSHPULL);
307. GpioDataRegs.GPBCLEAR.bit.GPIO61 = 1;
```

```
308.
309. // LED22
310. GPIO_SetupPinMux(157, GPIO_MUX_CPU1, 0);
311. GPIO_SetupPinOptions(157, GPIO_OUTPUT, GPIO_PUSHPULL);
312. GpioDataRegs.GPECLEAR.bit.GPIO157 = 1;
313.
314. // LED23
315. GPIO_SetupPinMux(158, GPIO_MUX_CPU1, 0);
316. GPIO_SetupPinOptions(158, GPIO_OUTPUT, GPIO_PUSHPULL);
317. GpioDataRegs.GPECLEAR.bit.GPIO158 = 1;
318.
319. //DRV8874 #1 DIR Direction
320. GPIO_SetupPinMux(29, GPIO_MUX_CPU1, 0);
321. GPIO_SetupPinOptions(29, GPIO_OUTPUT, GPIO_PUSHPULL);
322. GpioDataRegs.GPASET.bit.GPIO29 = 1;
323.
324. //DRV8874 #2 DIR Direction
325. GPIO_SetupPinMux(32, GPIO_MUX_CPU1, 0);
326. GPIO_SetupPinOptions(32, GPIO_OUTPUT, GPIO_PUSHPULL);
327. GpioDataRegs.GPBSET.bit.GPIO32 = 1;
328.
329. //MPU9250 CS Chip Select
330. GPIO_SetupPinMux(66, GPIO_MUX_CPU1, 0);
331. GPIO_SetupPinOptions(66, GPIO_OUTPUT, GPIO_PUSHPULL);
332. GpioDataRegs.GPCSET.bit.GPIO66 = 1;
333.
334. //PushButton 1
335. GPIO_SetupPinMux(122, GPIO_MUX_CPU1, 0);
336. GPIO_SetupPinOptions(122, GPIO_INPUT, GPIO_PULLUP);
337.
338. //PushButton 2
339. GPIO_SetupPinMux(123, GPIO_MUX_CPU1, 0);
340. GPIO_SetupPinOptions(123, GPIO_INPUT, GPIO_PULLUP);
341.
342. //PushButton 3
343. GPIO_SetupPinMux(124, GPIO_MUX_CPU1, 0);
344. GPIO_SetupPinOptions(124, GPIO_INPUT, GPIO_PULLUP);
345.
346. //PushButton 4
347. GPIO_SetupPinMux(125, GPIO_MUX_CPU1, 0);
348. GPIO_SetupPinOptions(125, GPIO_INPUT, GPIO_PULLUP);
349.
350. // control right motor input by PWM
351. GPIO_SetupPinMux(10, GPIO_MUX_CPU1, 1);
```

```

352.     GPIO_SetupPinOptions(10, GPIO_OUTPUT, GPIO_PUSHPULL);
353.     GpioDataRegs.GPACLEAR.bit.GPIO10 = 1;
354.
355.     // control left motor input by PWM
356.     GPIO_SetupPinMux(11, GPIO_MUX_CPU1, 1);
357.     GPIO_SetupPinOptions(11, GPIO_OUTPUT, GPIO_PUSHPULL);
358.     GpioDataRegs.GPACLEAR.bit.GPIO11 = 1;
359.
360.     // Clear all interrupts and initialize PIE vector table:
361.     // Disable CPU interrupts
362.     DINT;
363.
364.     setupSpib();
365.     init_eQEPs();
366.
367.     //disable pull-up resistor
368.     EALLOW; // Below are protected registers
369.     GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;
370.     GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;
371.     GpioCtrlRegs.GPAPUD.bit.GPIO22 = 1;
372.     GpioCtrlRegs.GPEPUD.bit.GPIO159 = 1;
373.     EDIS;
374.
375.     // Initialize the PIE control registers to their default state.
376.     // The default state is all PIE interrupts disabled and flags
377.     // are cleared.
378.     // This function is found in the F2837xD_PieCtrl.c file.
379.     InitPieCtrl();
380.
381.     // Disable CPU interrupts and clear all CPU interrupt flags:
382.     IER = 0x0000;
383.     IFR = 0x0000;
384.
385.     // Initialize the PIE vector table with pointers to the shell
        Interrupt
386.     // Service Routines (ISR).
387.     // This will populate the entire table, even if the interrupt
388.     // is not used in this example. This is useful for debug
        purposes.
389.     // The shell ISR routines are found in F2837xD_DefaultIsr.c.
390.     // This function is found in F2837xD_PieVect.c.
391.     InitPieVectTable();
392.
393.     // Interrupts that are used in this example are re-mapped to

```

```

394.    // ISR functions found within this project
395.    EALLOW; // This is needed to write to EALLOW protected
        registers
396.    PieVectTable.TIMER0_INT = &cpu_timer0_isr;
397.    PieVectTable.TIMER1_INT = &cpu_timer1_isr;
398.    PieVectTable.TIMER2_INT = &cpu_timer2_isr;
399.    PieVectTable.SCIA_RX_INT = &RXAINT_recv_ready;
400.    PieVectTable.SCIC_RX_INT = &RXCINT_recv_ready;
401.    PieVectTable.SCID_RX_INT = &RXDINT_recv_ready;
402.    PieVectTable.SCIA_TX_INT = &TXAINT_data_sent;
403.    PieVectTable.SCIC_TX_INT = &TXCINT_data_sent;
404.    PieVectTable.SCID_TX_INT = &TXDINT_data_sent;
405.    PieVectTable.ADCA1_INT = &ADCA_ISR;
406.    PieVectTable.SPIB_RX_INT = &SPIB_isr;
407.    PieVectTable.EMIF_ERROR_INT = &SWI_isr;
408.    EDIS; // This is needed to disable write to EALLOW protected
        registers
409.
410.    //EPWM5
411.    EALLOW;
412.    EPwm5Regs.ETSEL.bit.SOCAEN = 0; // Disable SOC on A group
413.    EPwm5Regs.TBCTL.bit.CTRMODE = 3; // freeze counter
414.    EPwm5Regs.ETSEL.bit.SOCASEL = 2; // Select Event when counter
        equal to PRD
415.    EPwm5Regs.ETPS.bit.SOCAPRD = 1; // Generate pulse on 1st event
        ("pulse" is the same as "trigger")
416.    EPwm5Regs.TBCTR = 0x0; // Clear counter
417.    EPwm5Regs.TBPHS.bit.TBPHS = 0x0000; // Phase is 0
418.    EPwm5Regs.TBCTL.bit.PHSEN = 0; // Disable phase loading
419.    EPwm5Regs.TBCTL.bit.CLKDIV = 0; // divide by 1 50Mhz Clock
420.    //ME461, Lab 3 Lab 5, Page 3 of 11
421.    EPwm5Regs.TBPRD = 50000; // Set Period to 1ms sample. Input
        clock is 50MHz.
422.    // Notice here that we are not setting CMPA or CMPB because we
        are not using the PWM signal
423.    EPwm5Regs.ETSEL.bit.SOCAEN = 1; //enable SOCA
424.    EPwm5Regs.TBCTL.bit.CTRMODE = 0; //unfreeze, and enter up
        count mode
425.    EDIS;
426.
427.    EALLOW;
428.
429.    //write configurations for all ADCs ADCA, ADCB, ADCC, ADCD
430.    AdcaRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4

```



```

431.     AdcbRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
432.     AdccRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
433.     AdcdRegs.ADCCTL2.bit.PRESCALE = 6; //set ADCCLK divider to /4
434.     AdcSetMode(ADC_ADCA, ADC_RESOLUTION_12BIT,
        ADC_SIGNALMODE_SINGLE); //read calibration settings
435.     AdcSetMode(ADC_ADCB, ADC_RESOLUTION_12BIT,
        ADC_SIGNALMODE_SINGLE); //read calibration settings
436.     AdcSetMode(ADC_ADCC, ADC_RESOLUTION_12BIT,
        ADC_SIGNALMODE_SINGLE); //read calibration settings
437.     AdcSetMode(ADC_ADCD, ADC_RESOLUTION_12BIT,
        ADC_SIGNALMODE_SINGLE); //read calibration settings
438.     //Set pulse positions to late
439.     AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
440.     AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
441.     AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
442.     AdcdRegs.ADCCTL1.bit.INTPULSEPOS = 1;
443.     //power up the ADCs
444.     AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
445.     AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
446.     AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
447.     AdcdRegs.ADCCTL1.bit.ADCPWDNZ = 1;
448.     //delay for 1ms to allow ADC time to power up
449.     DELAY_US(1000);
450.     //Select the channels to convert and end of conversion flag
451.     //Many statements commented out, To be used when using ADCA or
        ADCB
452.     //ME461, Lab 3 Lab 5, Page 4 of 11
453.
454.     //ADCA
455.     AdcaRegs.ADCSOC0CTL.bit.CHSEL = 2; //SOC0 will convert Channel
        you choose Does not have to be A0
456.     AdcaRegs.ADCSOC0CTL.bit.ACQPS = 14; //sample window is acqps + 1
        SYSCLK cycles = 75ns
457.     AdcaRegs.ADCSOC0CTL.bit.TRIGSEL = 0xD; // EPWM5 ADCSOCA or
        another trigger you choose will trigger SOC0
458.
459.     AdcaRegs.ADCSOC1CTL.bit.CHSEL = 3; //SOC1 will convert Channel
        you choose Does not have to be A1
460.     AdcaRegs.ADCSOC1CTL.bit.ACQPS = 14; //sample window is acqps + 1
        SYSCLK cycles = 75ns
461.     AdcaRegs.ADCSOC1CTL.bit.TRIGSEL = 0xD; // EPWM5 ADCSOCA or
        another trigger you choose will trigger SOC1
462.

```

```

463.     AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1; //set to last SOC that is
        converted and it will set INT1 flag ADCA1
464.     AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1; //enable INT1 flag
465.     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //make sure INT1 flag is
        cleared
466.     EDIS;
467.
468.     EALLOW;
469.     DacARegs.DACOUTEN.bit.DACOUTEN = 1; //enable dacA output-->uses
        ADCINA0
470.     DacARegs.DACCTL.bit.LOADMODE = 0; //load on next sysclk
471.     DacARegs.DACCTL.bit.DACREFSEL = 1; //use ADC VREF as reference
        voltage
472.     DacBRegs.DACOUTEN.bit.DACOUTEN = 1; //enable dacB output-->uses
        ADCINA1
473.     DacBRegs.DACCTL.bit.LOADMODE = 0; //load on next sysclk
474.     DacBRegs.DACCTL.bit.DACREFSEL = 1; //use ADC VREF as reference
        voltage
475.     EDIS;
476.
477.     // Initialize the CpuTimers Device Peripheral. This function can
        be
478.     // found in F2837xD_CpuTimers.c
479.     InitCpuTimers();
480.
481.     // Configure CPU-Timer 0, 1, and 2 to interrupt every second:
482.     // 200MHz CPU Freq, 1 second Period (in uSeconds)
483.     ConfigCpuTimer(&CpuTimer0, 200, 1000);
484.     ConfigCpuTimer(&CpuTimer1, 200, 20000);
485.     ConfigCpuTimer(&CpuTimer2, 200, 4000);
486.
487.     // Enable CpuTimer Interrupt bit TIE
488.     CpuTimer0Regs.TCR.all = 0x4000;
489.     CpuTimer1Regs.TCR.all = 0x4000;
490.     CpuTimer2Regs.TCR.all = 0x4000;
491.
492.     init_serial(&SerialA,115200,serialRXA);
493.     //     init_serial(&SerialC,115200,serialRXC);
494.     //     init_serial(&SerialD,115200,serialRXD);
495.
496.     // Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
497.     // which is connected to CPU-Timer 1, and CPU int 14, which is
        connected
498.     // to CPU-Timer 2: int 12 is for the SWI.

```

```

499.     IER |= M_INT1;
500.     IER |= M_INT8; // SCIC SCID
501.     IER |= M_INT9; // SCIA
502.     IER |= M_INT12;
503.     IER |= M_INT13;
504.     IER |= M_INT14;
505.     IER |= M_INT6; //SPIB_RX
506.
507.     //Enable ADCA1 in the PIE: Group 1 interrupt 1
508.     PieCtrlRegs.PIEIER1.bit.INTx1 = 1;
509.     // Enable SPIB_RX: Group 6 interrupt 3
510.     PieCtrlRegs.PIEIER6.bit.INTx3 = 1;
511.     // Enable TINT0 in the PIE: Group 1 interrupt 7
512.     PieCtrlRegs.PIEIER1.bit.INTx7 = 1;
513.     // Enable SWI in the PIE: Group 12 interrupt 9
514.     PieCtrlRegs.PIEIER12.bit.INTx9 = 1;
515.
516.     //initialize the PWM register 6A, 6B
517.     EPwm6Regs.TBCTL.bit.CTRMODE = 0;
518.     EPwm6Regs.TBCTL.bit.FREE_SOFT = 2;
519.     EPwm6Regs.TBCTL.bit.PHSEN = 0;
520.     EPwm6Regs.TBCTL.bit.CLKDIV = 0;
521.
522.     EPwm6Regs.TBCTR = 0;
523.     EPwm6Regs.TBPRD = 2500;
524.     EPwm6Regs.TBPHS.bit.TBPHS = 0;
525.     EPwm6Regs.CMPA.bit.CMPA = 0;
526.     EPwm6Regs.AQCTLA.bit.CAU = 1;
527.     EPwm6Regs.AQCTLA.bit.ZRO = 2;
528.
529.     EPwm6Regs.CMPB.bit.CMPB = 0;
530.     EPwm6Regs.AQCTLB.bit.CBU = 1;
531.     EPwm6Regs.AQCTLB.bit.ZRO = 2;
532.
533.     // Enable global Interrupts and higher priority real-time debug
events
534.     EINT; // Enable Global interrupt INTM
535.     ERTM; // Enable Global realtime interrupt DBGM
536.
537.     // IDLE loop. Just sit and loop forever (optional):
538.     while(1)
539.     {
540.         if (UARTPrint == 1 ) {

```

```

541.         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);
542.         UARTPrint = 0;
543.     }
544. }
545. }
546.
547.
548. // SWI_isr, Using this interrupt as a Software started interrupt
549. __interrupt void SWI_isr(void) {
550.
551.     // These three lines of code allow SWI_isr, to be interrupted by
other interrupt functions
552.     // making it lower priority than all other Hardware interrupts.
553.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP12;
554.     asm("        NOP");           // Wait one cycle
555.     EINT;                         // Clear INTM to enable
interrupts
556.
557.     // Insert SWI ISR Code here.....
558.
559.     turnref = turnref_1 + 0.002 * (turnrate + turnrate_1);
560.
561.     WhlDiff = LeftWheel - RightWheel;
562.     vel_WhlDiff = 0.333 * vel_WhlDiff_1 + 2.0/(3.0*0.004)*(WhlDiff -
WhlDiff_1);
563.     eDiff = turnref - WhlDiff;
564.     intDiff = intDiff_1 + 0.002 * (eDiff + eDiff_1);
565.
566.     turn = Kp*eDiff + Ki*intDiff - Kd*vel_WhlDiff;
567.
568.     if(fabs(turn)>3){
569.         intDiff = intDiff_1;
570.     }
571.     if(turn > 4.0){
572.         turn = 4.0;
573.     }
574.     if(turn < -4.0){
575.         turn = -4.0;
576.     }
577.
578.     WhlDiff_1 = WhlDiff;
579.     vel_WhlDiff_1 = vel_WhlDiff;

```

```

580.     eDiff_1 = eDiff;
581.     intDiff_1 = intDiff;
582.     turnref_1 = turnref;
583.     turnrate_1 = turnrate;
584.
585.     v_l = 0.6 * v_l_1 + 100 * (LeftWheel - LeftWheel_1);
586.     v_r = 0.6 * v_r_1 + 100 * (RightWheel - RightWheel_1);
587.
588.     Ubal = -K1 * tilt_value - K2 * gyro_value - K3 * (v_l +
        v_r)/2.0;
589.
590.     v_l_1 = v_l;
591.     v_r_1 = v_r;
592.
593.     uLeft = (Ubal/2.0) + turn + FwdBackOffset;
594.     uRight = (Ubal/2.0) - turn + FwdBackOffset;
595.
596.     setEPWM6A(uLeft);
597.     setEPWM6B(-uRight);
598.
599.     // Final Project add code here
600.     if (reset == 1) {
601.         xm = 0.0;
602.         ym = 0.0;
603.         xm_dot = 0.0;
604.         ym_dot = 0.0;
605.         xm_dot_1 = 0.0;
606.         ym_dot_1 = 0.0;
607.         theta_dot = 0.0;
608.         theta = 0.0;
609.         theta_1 = 0.0;
610.         phi = 0.0;
611.         theta_left = 0.0;
612.         theta_right = 0.0;
613.         reset = 0;
614.     }
615.     theta_left = theta_left + (LeftWheel - LeftWheel_1);
616.     theta_right = theta_right + (RightWheel - RightWheel_1);
617.     theta = 0.5 * (theta_left + theta_right);
618.     phi = Rw / W * (theta_right - theta_left);
619.     theta_dot = (theta - theta_1) / 0.004;
620.     xm_dot = Rw * theta_dot * cos(phi);
621.     ym_dot = Rw * theta_dot * sin(phi);
622.     xm = xm + 0.004 * (xm_dot + xm_dot_1);

```

```

623.     ym = ym + 0.004 * (ym_dot + ym_dot_1);
624.
625.     theta_1 = theta;
626.     xm_dot = xm_dot_1;
627.     ym_dot = ym_dot_1;
628.     LeftWheel_1 = LeftWheel;
629.     RightWheel_1 = RightWheel;
630.
631.     // Final project auto move part
632.     if (stage == 0) { // prepare stage
633.         phi_T = atan2(yT-ym,xT-xm); // specify the direction towards
        the target
634.     }
635.     if (stage == 1) { // turning stage
636.         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
637.             turnrate = 0.0;
638.             stage = 2;
639.         }
640.         else if (phi_T > 0) {
641.             turnrate = -1.0; // turn left to face the target
642.         }
643.         else if (phi_T < 0) {
644.             turnrate = 1.0; // turn right to face the target
645.         }
646.     }
647.     if (stage == 2) { // moving stage
648.         if ((fabs(xm-xT) < 0.01) && (fabs(ym-yT) < 0.01)) {
649.             turnrate = 0.0;
650.             FwdBackOffset = 0.0;
651.             stage = 3;
652.         }
653.         else {
654.             FwdBackOffset = -1.5;
655.         }
656.     }
657.
658.     numSWIcalls++;
659.     DINT;
660. }
661.
662. // cpu_timer0_isr - CPU Timer0 ISR
663. __interrupt void cpu_timer0_isr(void)
664. {

```

```

665.     CpuTimer0.InterruptCount++;
666.
667.     numTimer0calls++;
668.
669. //     if ((numTimer0calls%50) == 0) {
670. //         PieCtrlRegs.PIEIFR12.bit.INTx9 = 1; // Manually cause the
        interrupt for the SWI
671. //     }
672.
673.
674.     // Blink LaunchPad Red LED
675.     //GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
676.     // Blink a number of LEDs
677.     // GpioDataRegs.GPATOGGLE.bit.GPIO7 = 1;
678.     // GpioDataRegs.GPATOGGLE.bit.GPIO8 = 1;
679.     // GpioDataRegs.GPATOGGLE.bit.GPIO9 = 1;
680.     // GpioDataRegs.GPATOGGLE.bit.GPIO24 = 1;
681.     // GpioDataRegs.GPATOGGLE.bit.GPIO25 = 1;
682.     // GpioDataRegs.GPATOGGLE.bit.GPIO26 = 1;
683.
684.     // Acknowledge this interrupt to receive more interrupts from
        group 1
685.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
686. }
687.
688. // cpu_timer1_isr - CPU Timer1 ISR
689. __interrupt void cpu_timer1_isr(void)
690. {
691.
692.
693.     // Blink a number of LEDs
694.     // GpioDataRegs.GPATOGGLE.bit.GPIO22 = 1;
695.     // GpioDataRegs.GPBTOGGLE.bit.GPIO52 = 1;
696.     // GpioDataRegs.GPCTOGGLE.bit.GPIO67 = 1;
697.     // GpioDataRegs.GPCTOGGLE.bit.GPIO94 = 1;
698.     // GpioDataRegs.GPCTOGGLE.bit.GPIO95 = 1;
699.     // GpioDataRegs.GPDTOGGLE.bit.GPIO97 = 1;
700.
701.     CpuTimer1.InterruptCount++;
702. }
703.
704. // cpu_timer2_isr CPU Timer2 ISR
705. __interrupt void cpu_timer2_isr(void)
706. {

```

```

707.
708.
709.    // Blink LaunchPad Blue LED
710.    //GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1;
711.    // Blink a number of Leds
712.    //    GpioDataRegs.GPDTOGGLE.bit.GPIO111 = 1;
713.    //    GpioDataRegs.GPETOGGLE.bit.GPIO130 = 1;
714.    //    GpioDataRegs.GPETOGGLE.bit.GPIO131 = 1;
715.    //    GpioDataRegs.GPATOGGLE.bit.GPIO4 = 1;
716.    //    GpioDataRegs.GPATOGGLE.bit.GPIO5 = 1;
717.    //    GpioDataRegs.GPATOGGLE.bit.GPIO6 = 1;
718. //    CpuTimer2.InterruptCount++;
719. //        if ((CpuTimer2.InterruptCount % 50) == 0) {
720. //            UARTPrint = 1;
721. //        }
722.
723.
724. }
725.
726.
727. //b is the filter coefficients
728. float b[10]={    1.1982297073578186e-02,
729.                 3.2593697188218529e-02,
730.                 8.8809724362308426e-02,
731.                 1.5903360855022139e-01,
732.                 2.0758067282567344e-01,
733.                 2.0758067282567344e-01,
734.                 1.5903360855022139e-01,
735.                 8.8809724362308426e-02,
736.                 3.2593697188218529e-02,
737.                 1.1982297073578186e-02};
738.
739. //xk is the current ADC reading, xk_1 is the ADC reading one
    millisecond ago, xk_2 two milliseconds ago, etc
740. float xk[10];
741. float zk[10];
742.
743. float yk = 0.0;
744. float wk = 0.0;
745.
746. uint32_t adca0result = 0;
747. uint32_t adca1result = 0;
748.
749. //adca1 pie interrupt

```



```

750. __interrupt void ADCA_ISR (void)
751. {
752.     int i = 0;
753.     numADCACalls ++;
754.
755.     //save adc output
756.     adca0result = AdcaResultRegs.ADCRESULT0;
757.     adcalresult = AdcaResultRegs.ADCRESULT1;
758.
759.     // Here covert ADCIND0, ADCIND1 to volts
760.     analogvalue0 = 3.0*(adca0result / 4095.0);
761.     analogvalue1 = 3.0*(adcalresult / 4095.0);
762.
763.     xk[0] = analogvalue0;
764.
765.     for(i=0; i<=9; i++){
766.         yk += b[i]*xk[i];
767.     }
768.     out1 = yk;
769.     yk = 0.0;
770.
771.     //Save past states before exiting from the function so that next
       sample they are the older state
772.     for(i=9; i>0; i--){
773.         xk[i] = xk[i-1];
774.     }
775.
776.     zk[0] = analogvalue1;
777.
778.     for(i=0; i<=9; i++){
779.         wk += b[i]*zk[i];
780.     }
781.     out2 = wk;
782.     wk = 0.0;
783.
784.     //Save past states before exiting from the function so that next
       sample they are the older state
785.     for(i=9; i>0; i--){
786.         zk[i] = zk[i-1];
787.     }
788.
789.
790.     // Code inside CPU Timer 0
791.     GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;

```

```

792.     SpibRegs.SPIFFRX.bit.RXFFIL = 8;
793.     SpibRegs.SPITXBUF = ((0x8000)|(0x3A00)); //send random value to
    the former address
794.     SpibRegs.SPITXBUF = 0; //to receive x_acceleration
795.     SpibRegs.SPITXBUF = 0; //to receive y_acceleration
796.     SpibRegs.SPITXBUF = 0; //to receive z_acceleration
797.     SpibRegs.SPITXBUF = 0; //receive temp
798.     SpibRegs.SPITXBUF = 0; //to receive x_gyro
799.     SpibRegs.SPITXBUF = 0; //to receive y_gyro
800.     SpibRegs.SPITXBUF = 0; //to receive z_gyro
801.
802. //      // Print ADCIND0 and ADCIND1's voltage value to TeraTerm every
    100ms
803. //      if ((numADCacalls % 100) == 0) {
804. //          UARTPrint = 1;
805. //      }
806.
807.     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1; //clear interrupt flag
808.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
809.
810. }
811.
812. __interrupt void SPIB_isr(void){
813.
814.     int16_t temp1 = 0;
815.     numSPICalls++;
816.
817.     temp1 = SpibRegs.SPIRXBUF;
818.     x_accel = SpibRegs.SPIRXBUF;
819.     y_accel = SpibRegs.SPIRXBUF;
820.     z_accel = SpibRegs.SPIRXBUF;
821.     temp1 = SpibRegs.SPIRXBUF;
822.     x_gyro = SpibRegs.SPIRXBUF;
823.     y_gyro = SpibRegs.SPIRXBUF;
824.     z_gyro = SpibRegs.SPIRXBUF;
825.     //
826.     GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Set GPIO 66 to end Slave
    Select of MPU.
827.
828.     gyrox = x_gyro * 250.0 / 32767.0;
829.     gyroy = y_gyro * 250.0 / 32767.0;
830.     gyroz = z_gyro * 250.0 / 32767.0;
831.     accelx = x_accel * 4.0 / 32767.0;
832.     accely = y_accel * 4.0 / 32767.0;

```

```

833.     accelz = z_accel * 4.0 / 32767.0;
834.
835.
836.     //Code to be copied into SPIB_ISR interrupt function after the
      IMU measurements have been collected.
837.     if(calibration_state == 0){
838.         calibration_count++;
839.         if (calibration_count == 2000) {
840.             calibration_state = 1;
841.             calibration_count = 0;
842.         }
843.     }
844.
845.     else if(calibration_state == 1){
846.         accelx_offset+=accelx;
847.         accely_offset+=accely;
848.         accelz_offset+=accelz;
849.         gyrox_offset+=gyrox;
850.         gyroy_offset+=gyroy;
851.         gyroz_offset+=gyroz;
852.         calibration_count++;
853.         if (calibration_count == 2000) {
854.             calibration_state = 2;
855.             accelx_offset/=2000.0;
856.             accely_offset/=2000.0;
857.             accelz_offset/=2000.0;
858.             gyrox_offset/=2000.0;
859.             gyroy_offset/=2000.0;
860.             gyroz_offset/=2000.0;
861.             calibration_count = 0;
862.             doneCal = 1;
863.         }
864.     }
865.
866.     else if(calibration_state == 2){
867.         accelx -=(accelx_offset);
868.         accely -=(accely_offset);
869.         accelz -=(accelz_offset-accelzBalancePoint);
870.         gyrox -= gyrox_offset;
871.         gyroy -= gyroy_offset;
872.         gyroz -= gyroz_offset;
873.
874.         /*-----Kalman Filtering code start-----
      -----*/

```

```

875.         float tiltrate = (gyrox*PI)/180.0; // rad/s
876.         float pred_tilt, z, y, S;
877.
878.         tilt_rate = tiltrate;
879.
880.         // Prediction Step
881.         pred_tilt = kalman_tilt + T*tiltrate;
882.         pred_P = kalman_P + Q;
883.
884.         // Update Step
885.         z = -accelz;
886.         y = z - pred_tilt;
887.         S = pred_P + R;
888.         kalman_K = pred_P/S;
889.         kalman_tilt = pred_tilt + kalman_K*y;
890.         kalman_P = (1 - kalman_K)*pred_P;
891.         SpibNumCalls++;
892.
893.         // Kalman Filter used
894.         tilt_array[SpibNumCalls] = kalman_tilt;
895.         gyro_array[SpibNumCalls] = tiltrate;
896.         LeftWheelArray[SpibNumCalls] = readEncLeft();
897.         RightWheelArray[SpibNumCalls] = -readEncRight();
898.
899.         if (SpibNumCalls >= 3) { // should never be greater than 3
900.             tilt_value = (tilt_array[0] + tilt_array[1] +
                tilt_array[2] + tilt_array[3])/4.0;
901.             gyro_value = (gyro_array[0] + gyro_array[1] +
                gyro_array[2] + gyro_array[3])/4.0;
902.
903.             LeftWheel=(LeftWheelArray[0]+LeftWheelArray[1]+LeftWheelArray[2]+Left
                WheelArray[3])/4.0;
904.             RightWheel=(RightWheelArray[0]+RightWheelArray[1]+RightWheelArray[2]+
                RightWheelArray[3])/4.0;
905.             SpibNumCalls = -1;
906.             PieCtrlRegs.PIEIFR12.bit.INTx9 = 1; // Manually cause the
                interrupt for the SWI
907.         }
908.
909.         timecount++;
910.         if((timecount%200) == 0)
911.         {

```

```

912.         if(doneCal == 0) {
913.             GpioDataRegs.GPATOGGLE.bit.GPIO31 = 1; // Blink Blue LED
               while calibrating
914.         }
915.         GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1; // Always Block Red
               LED
916.         UARTPrint = 1; // Tell While loop to print
917.     }
918.
919.     SpibRegs.SPIFFRX.bit.RXFFOVFCLR=1; // Clear Overflow flag
920.     SpibRegs.SPIFFRX.bit.RXFFINTCLR=1; // Clear Interrupt flag
921.     PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;
922.
923. }
924.
925. // This function is called each time a char is recieved over UARTA.
926. //use WASD to control sebot moving
927. void serialRXA(serial_t *s, char data) {
928.     numRXA ++;
929.     if (data == 'a') { // turn left
930.         turnrate = turnrate - 0.2;
931.     } else if (data == 'd') { // turn right
932.         turnrate = turnrate + 0.2;
933.     } else if (data == 'w') { // go forward
934.         FwdBackOffset = FwdBackOffset - 0.2;
935.     } else if (data == 's') { // go backward
936.         FwdBackOffset = FwdBackOffset + 0.2;
937.     } else if (data == 'r') { // reset position
938.         turnrate = 0;
939.         FwdBackOffset = 0;
940.         reset = 1;
941.     } else if (data == 't') { // start auto moving
942.         stage = 1;
943.     } else { // reset speed
944.         turnrate = 0;
945.         FwdBackOffset = 0;
946.     }
947. }
948.
949.
950. void setupSpib(void) //Call this function in main() somewhere after
               the DINT; line of code.
951. {
952.     int16_t temp = 0;

```

```

953.     int16_t i_temp = 0;
954.     //Step 1.
955.     // cut and paste here all the SpibRegs initializations you found
        for part 3. Change so that 16 bits are
956.     //transmitted each TX FIFO write and change the delay in between
        each transfer to 0.
957.     //-----
        -----
958.
959.
960.     GPIO_SetupPinMux(2, GPIO_MUX_CPU1, 0); // Set as GPIO2 and used
        as DAN777 SS
961.     GPIO_SetupPinOptions(2, GPIO_OUTPUT, GPIO_PUSHPULL); // Make
        GPIO2 an Output Pin
962.     GpioDataRegs.GPASET.bit.GPIO2 = 1; //Initially Set GPIO2/SS High
        so DAN777 is not selected
963.
964.     GPIO_SetupPinMux(66, GPIO_MUX_CPU1, 0); // Set as GPIO66 and
        used as MPU-9250 SS
965.     GPIO_SetupPinOptions(66, GPIO_OUTPUT, GPIO_PUSHPULL); // Make
        GPIO66 an Output Pin
966.     GpioDataRegs.GPCSET.bit.GPIO66 = 1; //Initially Set GPIO66/SS
        High so MPU-9250 is not selected
967.
968.     GPIO_SetupPinMux(63, GPIO_MUX_CPU1, 15); //Set GPIO63 pin to
        SPISIMOB
969.     GPIO_SetupPinMux(64, GPIO_MUX_CPU1, 15); //Set GPIO64 pin to
        SPISOMIB
970.     GPIO_SetupPinMux(65, GPIO_MUX_CPU1, 15); //Set GPIO65 pin to
        SPICLKB
971.
972.     EALLOW;
973.     GpioCtrlRegs.GPBPUD.bit.GPIO63 = 0; // Enable Pull-ups on SPI
        PINS Recommended by TI for SPI Pins
974.     GpioCtrlRegs.GPCPUD.bit.GPIO64 = 0;
975.     GpioCtrlRegs.GPCPUD.bit.GPIO65 = 0;
976.     GpioCtrlRegs.GPBQSEL2.bit.GPIO63 = 3; // Set prequalifier for
        SPI PINS
977.     GpioCtrlRegs.GPCQSEL1.bit.GPIO64 = 3; // The prequalifier
        eliminates short noise spikes
978.     GpioCtrlRegs.GPCQSEL1.bit.GPIO65 = 3; // by making sure the
        serial pin stays low for 3 clock periods.
979.     EDIS;
980.     SpibRegs.SPICCR.bit.SPISWRESET = 0; // Put SPI in Reset

```

```

981.     SpibRegs.SPICTL.bit.CLK_PHASE = 1; //This happens to be the mode
        for both the DAN777 and
982.     SpibRegs.SPICCR.bit.CLKPOLARITY = 0; //The MPU-9250, Mode 01.
983.     SpibRegs.SPICTL.bit.MASTER_SLAVE = 1; // Set to SPI Master
984.     SpibRegs.SPICCR.bit.SPICHAR = 0xF; // Set to transmit and
        receive 16 bits each write to SPITXBUF
985.     SpibRegs.SPICTL.bit.TALK = 1; // Enable transmission
986.     SpibRegs.SPIPRI.bit.FREE = 1; // Free run, continue SPI
        operation
987.     SpibRegs.SPICTL.bit.SPIINTENA = 0; // Disables the SPI interrupt
988.     SpibRegs.SPIBRR.bit.SPI_BIT_RATE = 49; // Set SCLK bit rate to 1
        MHz so 1us period. SPI base clock is
989.     // 50MHZ. And this setting divides that base clock to create
        SCLK's period
990.     SpibRegs.SPISTS.all = 0x0000; // Clear status flags just in case
        they are set for some reason
991.     SpibRegs.SPIFFTX.bit.SPIRST = 1; // Pull SPI FIFO out of reset,
        SPI FIFO can resume transmit or receive.
992.     SpibRegs.SPIFFTX.bit.SPIFFENA = 1; // Enable SPI FIFO
        enhancements
993.     SpibRegs.SPIFFTX.bit.TXFIFO = 0; // Write 0 to reset the FIFO
        pointer to zero, and hold in reset
994.     SpibRegs.SPIFFTX.bit.TXFFINTCLR = 1; // Write 1 to clear
        SPIFFTX[TXFFINT] flag just in case it is set
995.     SpibRegs.SPIFFRX.bit.RXFIFORESET = 0; // Write 0 to reset the
        FIFO pointer to zero, and hold in reset
996.     SpibRegs.SPIFFRX.bit.RXFFOVFCLR = 1; // Write 1 to clear
        SPIFFRX[RXFFOVF] just in case it is set
997.     SpibRegs.SPIFFRX.bit.RXFFINTCLR = 1; // Write 1 to clear
        SPIFFRX[RXFFINT] flag just in case it is set
998.     SpibRegs.SPIFFRX.bit.RXFFIENA = 1; // Enable the RX FIFO
        Interrupt. RXFFST >= RXFFIL
999.     SpibRegs.SPIFFCT.bit.TXDLY = 0x00; //Set delay between transmits
        to 16 spi clocks. Needed by DAN777 chip
1000.     SpibRegs.SPICCR.bit.SPISWRESET = 1; // Pull the SPI out
        of reset
1001.     SpibRegs.SPIFFTX.bit.TXFIFO = 1; // Release transmit
        FIFO from reset.
1002.     SpibRegs.SPIFFRX.bit.RXFIFORESET = 1; // Re-enable
        receive FIFO operation
1003.     SpibRegs.SPICTL.bit.SPIINTENA = 1; // Enables SPI
        interrupt. !! I don't think this is needed. Need to Test
1004.     SpibRegs.SPIFFRX.bit.RXFFIL = 2; //Interrupt Level to 2
        words or more received into FIFO causes interrupt

```

```

1005.
1006.
1007.          //Step 2.
1008.          // perform a multiple 16 bit transfer to initialize MPU-
          9250 registers 0x13,0x14,0x15,0x16
1009.          // 0x17, 0x18, 0x19, 0x1A, 0x1B, 0x1C 0x1D, 0x1E, 0x1F.
          Use only one SS low to high for all these writes
1010.          // some code is given, most you have to fill you
          yourself.
1011.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1; // Slave Select
          Low
1012.          // Perform the number of needed writes to SPITXBUF to
          write to all 13 registers
1013.
1014.          // To address 00x13 write 0x00
1015.          SpibRegs.SPITXBUF = 0x1300;
1016.          // To address 00x14 write 0x00
1017.          // To address 00x15 write 0x00
1018.          SpibRegs.SPITXBUF = 0x0;
1019.          // To address 00x16 write 0x00
1020.          // To address 00x17 write 0x00
1021.          SpibRegs.SPITXBUF = 0x0;
1022.          // To address 00x18 write 0x00
1023.          // To address 00x19 write 0x13
1024.          SpibRegs.SPITXBUF = 0x0013;
1025.          // To address 00x1A write 0x02
1026.          // To address 00x1B write 0x00
1027.          SpibRegs.SPITXBUF = 0x0200;
1028.          // To address 00x1C write 0x08
1029.          // To address 00x1D write 0x06
1030.          SpibRegs.SPITXBUF = 0x0806;
1031.          // To address 00x1E write 0x00
1032.          // To address 00x1F write 0x00
1033.          SpibRegs.SPITXBUF = 0x0;
1034.
1035.          // wait for the correct number of 16 bit values to be
          received into the RX FIFO
1036.          while(SpibRegs.SPIFFRX.bit.RXFFST != 7);
1037.          GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Slave Select High
1038.          for(i_temp=0; i_temp<7; i_temp++){
1039.              temp = SpibRegs.SPIRXBUF;
1040.          }
1041.          // ??? read the additional number of garbage receive
          values off the RX FIFO to clear out the RX FIFO

```



```

1042.          DELAY_US(10); // Delay 10us to allow time for the MPU-
          2950 to get ready for next transfer.
1043.
1044.
1045.          //Step 3.
1046.          // perform a multiple 16 bit transfer to initialize MPU-
          9250 registers 0x23,0x24,0x25,0x26
1047.          // 0x27, 0x28, 0x29. Use only one SS low to high for all
          these writes
1048.          // some code is given, most you have to fill you
          yourself.
1049.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1; // Slave Select
          Low
1050.          // Perform the number of needed writes to SPITXBUF to
          write to all 13 registers
1051.          // To address 00x23 write 0x00
1052.          SpibRegs.SPITXBUF = 0x2300;
1053.          // To address 00x24 write 0x40
1054.          // To address 00x25 write 0x8C
1055.          SpibRegs.SPITXBUF = 0x408C;
1056.          // To address 00x26 write 0x02
1057.          // To address 00x27 write 0x88
1058.          SpibRegs.SPITXBUF = 0x0288;
1059.          // To address 00x28 write 0x0C
1060.          // To address 00x29 write 0x0A
1061.          SpibRegs.SPITXBUF = 0x0C0A;
1062.
1063.
1064.          // wait for the correct number of 16 bit values to be
          received into the RX FIFO
1065.          while(SpibRegs.SPIFFRX.bit.RXFFST != 4);
1066.          GpioDataRegs.GPCSET.bit.GPIO66 = 1; // Slave Select High
1067.          temp = SpibRegs.SPIRXBUF;
1068.          temp = SpibRegs.SPIRXBUF;
1069.          temp = SpibRegs.SPIRXBUF;
1070.          temp = SpibRegs.SPIRXBUF;
1071.          // read the additional number of garbage receive
          values off the RX FIFO to clear out the RX FIFO
1072.          DELAY_US(10); // Delay 10us to allow time for the MPU-
          2950 to get ready for next transfer.
1073.
1074.
1075.          //Step 4.

```

```

1076.          // perform a single 16 bit transfer to initialize MPU-
          9250 register 0x2A
1077.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1078.          // Write to address 0x2A the value 0x81
1079.          SpibRegs.SPITXBUF = 0x2A81;
1080.          // wait for one byte to be received
1081.          while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1082.          GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1083.          temp = SpibRegs.SPIRXBUF;
1084.          DELAY_US(10);
1085.
1086.          // The Remainder of this code is given to you and you do
          not need to make any changes.
1087.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1088.          SpibRegs.SPITXBUF = (0x3800 | 0x0001); // 0x3800
1089.          while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1090.          GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1091.          temp = SpibRegs.SPIRXBUF;
1092.          DELAY_US(10);
1093.
1094.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1095.          SpibRegs.SPITXBUF = (0x3A00 | 0x0001); // 0x3A00
1096.          while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1097.          GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1098.          temp = SpibRegs.SPIRXBUF;
1099.          DELAY_US(10);
1100.
1101.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1102.          SpibRegs.SPITXBUF = (0x6400 | 0x0001); // 0x6400
1103.          while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1104.          GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1105.          temp = SpibRegs.SPIRXBUF;
1106.          DELAY_US(10);
1107.
1108.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1109.          SpibRegs.SPITXBUF = (0x6700 | 0x0003); // 0x6700
1110.          while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1111.          GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1112.          temp = SpibRegs.SPIRXBUF;
1113.          DELAY_US(10);
1114.
1115.          GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1116.          SpibRegs.SPITXBUF = (0x6A00 | 0x0020); // 0x6A00
1117.          while(SpibRegs.SPIFFRX.bit.RXFFST !=1);

```

```
1118.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1119.         temp = SpibRegs.SPIRXBUF;
1120.         DELAY_US(10);
1121.
1122.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1123.         SpibRegs.SPITXBUF = (0x6B00 | 0x0001); // 0x6B00
1124.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1125.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1126.         temp = SpibRegs.SPIRXBUF;
1127.         DELAY_US(10);
1128.
1129.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1130.         SpibRegs.SPITXBUF = (0x7500 | 0x0071); // 0x7500
1131.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1132.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1133.         temp = SpibRegs.SPIRXBUF;
1134.         DELAY_US(10);
1135.
1136.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1137.         SpibRegs.SPITXBUF = (0x7700 | 0x00EB); // 0x7700
1138.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1139.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1140.         temp = SpibRegs.SPIRXBUF;
1141.         DELAY_US(10);
1142.
1143.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1144.         SpibRegs.SPITXBUF = (0x7800 | 0x0012); // 0x7800
1145.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1146.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1147.         temp = SpibRegs.SPIRXBUF;
1148.         DELAY_US(10);
1149.
1150.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1151.         SpibRegs.SPITXBUF = (0x7A00 | 0x0010); // 0x7A00
1152.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1153.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1154.         temp = SpibRegs.SPIRXBUF;
1155.         DELAY_US(10);
1156.
1157.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1158.         SpibRegs.SPITXBUF = (0x7B00 | 0x00FA); // 0x7B00
1159.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1160.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1161.         temp = SpibRegs.SPIRXBUF;
```

```

1162.         DELAY_US(10);
1163.
1164.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1165.         SpibRegs.SPITXBUF = (0x7D00 | 0x0021); // 0x7D00
1166.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1167.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1168.         temp = SpibRegs.SPIRXBUF;
1169.         DELAY_US(10);
1170.
1171.         GpioDataRegs.GPCCLEAR.bit.GPIO66 = 1;
1172.         SpibRegs.SPITXBUF = (0x7E00 | 0x0050); // 0x7E00
1173.         while(SpibRegs.SPIFFRX.bit.RXFFST !=1);
1174.         GpioDataRegs.GPCSET.bit.GPIO66 = 1;
1175.         temp = SpibRegs.SPIRXBUF;
1176.         DELAY_US(50);
1177.
1178.         // Clear SPIB interrupt source just in case it was
        issued due to any of the above initializations.
1179.         SpibRegs.SPIFFRX.bit.RXFFOVFCLR=1; // Clear Overflow
        flag
1180.         SpibRegs.SPIFFRX.bit.RXFFINTCLR=1; // Clear Interrupt
        flag
1181.         PieCtrlRegs.PIEACK.all = PIEACK_GROUP6;
1182.     }
1183.
1184.
1185.
1186.     void init_eQEPs(void) {
1187.         // setup eQEP1 pins for input
1188.         EALLOW;
1189.         //Disable internal pull-up for the selected output pins
        for reduced power consumption
1190.         GpioCtrlRegs.GPAPUD.bit.GPIO20 = 1; // Disable pull-up
        on GPIO20 (EQEP1A)
1191.         GpioCtrlRegs.GPAPUD.bit.GPIO21 = 1; // Disable pull-up
        on GPIO21 (EQEP1B)
1192.         GpioCtrlRegs.GPAQSEL2.bit.GPIO20 = 2; // Qual every 6
        samples
1193.         GpioCtrlRegs.GPAQSEL2.bit.GPIO21 = 2; // Qual every 6
        samples
1194.         EDIS;
1195.         // This specifies which of the possible GPIO pins will
        be EQEP1 functional pins.
1196.         // Comment out other unwanted lines.

```

```

1197.         GPIO_SetupPinMux(20, GPIO_MUX_CPU1, 1);
1198.         GPIO_SetupPinMux(21, GPIO_MUX_CPU1, 1);
1199.         EQep1Regs.QEPCTL.bit.QPEN = 0; // make sure eqep in
        reset
1200.         EQep1Regs.QDECCTL.bit.QSRC = 0; // Quadrature count mode
1201.         EQep1Regs.QPOSCTL.all = 0x0; // Disable eQep Position
        Compare
1202.         EQep1Regs.QCAPCTL.all = 0x0; // Disable eQep Capture
1203.         EQep1Regs.QEINT.all = 0x0; // Disable all eQep
        interrupts
1204.         EQep1Regs.QPOSMAX = 0xFFFFFFFF; // use full range of the
        32 bit count
1205.         EQep1Regs.QEPCTL.bit.FREE_SOFT = 2; // EQep uneffected
        by emulation suspend in Code Composer
1206.         EQep1Regs.QEPCTL.bit.QPEN = 1; // Enable EQep
1207.         EQep1Regs.QPOSCNT = 0;
1208.         // setup QEP2 pins for input
1209.         EALLOW;
1210.         //Disable internal pull-up for the selected output
        pinsfor reduced power consumption
1211.         GpioCtrlRegs.GPBPUD.bit.GPIO54 = 1; // Disable pull-up
        on GPIO54 (EQEP2A)
1212.         GpioCtrlRegs.GPBPUD.bit.GPIO55 = 1; // Disable pull-up
        on GPIO55 (EQEP2B)
1213.         GpioCtrlRegs.GPBQSEL2.bit.GPIO54 = 2; // Qual every 6
        samples
1214.         GpioCtrlRegs.GPBQSEL2.bit.GPIO55 = 2; // Qual every 6
        samples
1215.         EDIS;
1216.         GPIO_SetupPinMux(54, GPIO_MUX_CPU1, 5); // set GPIO54
        and eQep2A
1217.         GPIO_SetupPinMux(55, GPIO_MUX_CPU1, 5); // set GPIO54
        and eQep2B
1218.         EQep2Regs.QEPCTL.bit.QPEN = 0; // make sure qep reset
1219.         EQep2Regs.QDECCTL.bit.QSRC = 0; // Quadrature count mode
1220.         EQep2Regs.QPOSCTL.all = 0x0; // Disable eQep Position
        Compare
1221.         EQep2Regs.QCAPCTL.all = 0x0; // Disable eQep Capture
1222.         EQep2Regs.QEINT.all = 0x0; // Disable all eQep
        interrupts
1223.         EQep2Regs.QPOSMAX = 0xFFFFFFFF; // use full range of the
        32 bit count.
1224.         EQep2Regs.QEPCTL.bit.FREE_SOFT = 2; // EQep uneffected
        by emulation suspend

```

```

1225.         EQep2Regs.QEPCTL.bit.QPEN = 1; // Enable EQep
1226.         EQep2Regs.QPOSCNT = 0;
1227.     }
1228.
1229.     float readEncLeft(void) {
1230.         int32_t raw = 0;
1231.         uint32_t QEP_maxvalue = 0xFFFFFFFFU; //4294967295U
1232.         raw = EQep1Regs.QPOSCNT;
1233.         if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue; // I
            don't think this is needed and never true
1234.         // 20 North South magnet poles in the encoder disk so 20
            square waves per one revolution of the
1235.         // DC motor's back shaft. Then Quadrature Decoder mode
            multiplies this by 4 so 80 counts per one rev
1236.         // of the DC motor's back shaft. Then the gear motor's
            gear ratio is 18.7.
1237.         return (raw*(2*PI/80.0)/18.7);
1238.     }
1239.
1240.     float readEncRight(void) {
1241.         int32_t raw = 0;
1242.         uint32_t QEP_maxvalue = 0xFFFFFFFFU; //4294967295U -1
            32bit signed int
1243.         raw = EQep2Regs.QPOSCNT;
1244.         if (raw >= QEP_maxvalue/2) raw -= QEP_maxvalue; // I
            don't think this is needed and never true
1245.         // 20 North South magnet poles in the encoder disk so 20
            square waves per one revolution of the
1246.         // DC motor's back shaft. Then Quadrature Decoder mode
            multiplies this by 4 so 80 counts per one rev
1247.         // of the DC motor's back shaft. Then the gear motor's
            gear ratio is 18.7.
1248.         return (raw*(2*PI/80.0)/18.7);
1249.     }
1250.
1251.     void setEPWM6A (float controleffort){
1252.
1253.         if (controleffort >= 10){
1254.             controleffort = 10;
1255.         }
1256.         else if (controleffort <= -10){
1257.             controleffort = -10;
1258.         }
1259.

```

```
1260.         if (controleffort >= 0){
1261.             GpioDataRegs.GPASET.bit.GPIO29 = 1;
1262.         }
1263.         else{
1264.             GpioDataRegs.GPACLEAR.bit.GPIO29 = 1;
1265.         }
1266.         EPwm6Regs.CMPA.bit.CMPA = 2500 * (fabs(controleffort) /
1267.         10.0);
1268.     }
1269.     void setEPWM6B (float controleffort){
1270.
1271.         if (controleffort >= 10){
1272.             controleffort = 10;
1273.
1274.         }
1275.         else if (controleffort <= -10){
1276.             controleffort = -10;
1277.         }
1278.
1279.         if (controleffort >= 0){
1280.             GpioDataRegs.GPBSET.bit.GPIO32 = 1;
1281.         }
1282.         else{
1283.             GpioDataRegs.GPBCLEAR.bit.GPIO32 = 1;
1284.         }
1285.         EPwm6Regs.CMPB.bit.CMPB = 2500 * (fabs(controleffort) /
1286.         10.0);
1287.     }
```