
# CECS 347 Spring 2022 Project # 2

# An Autonomous Wall Follower Robot Car

By

Dylan Ramos & Jason Jingco

April 18, 2022

An Autonomous Wall Follower Robot Car Model robot car that can navigate through a prebuilt track with walls on both sides.

**Introduction**

This project was an accumulation of several previous projects and labs. For example, ADC GPIOs, interrupts, pulse width modulation (PWM), phased locked loop(PLL), power supply circuits, motor drivers, and DC motors. The purpose of this lab is to create a model car that passes through a track based on the ADC IR sensors to change the direction of the car.

**Operation**

The LEDs will indicate the current motion state of the car with yellow indicating the start of the car. When the LED is purple the car has reached the end of the track and it is in an open space. The first switch (SW1 or PF0) controls the start and stop of the car. Initially, the car will start static and will continue to move if there is a track barrier within 60 cm of each side. Switch 2 (SW2 or PF4) confirms the speed choice via the potentiometer which sets the speed within a specific range. If the LED is blue or green then the car is too close to the left or right side of the track and will correct itself. If the LED is ever red the car has collided into the wall. Lastly, reset will restart the state back to no motion.

Video links to the demonstration:

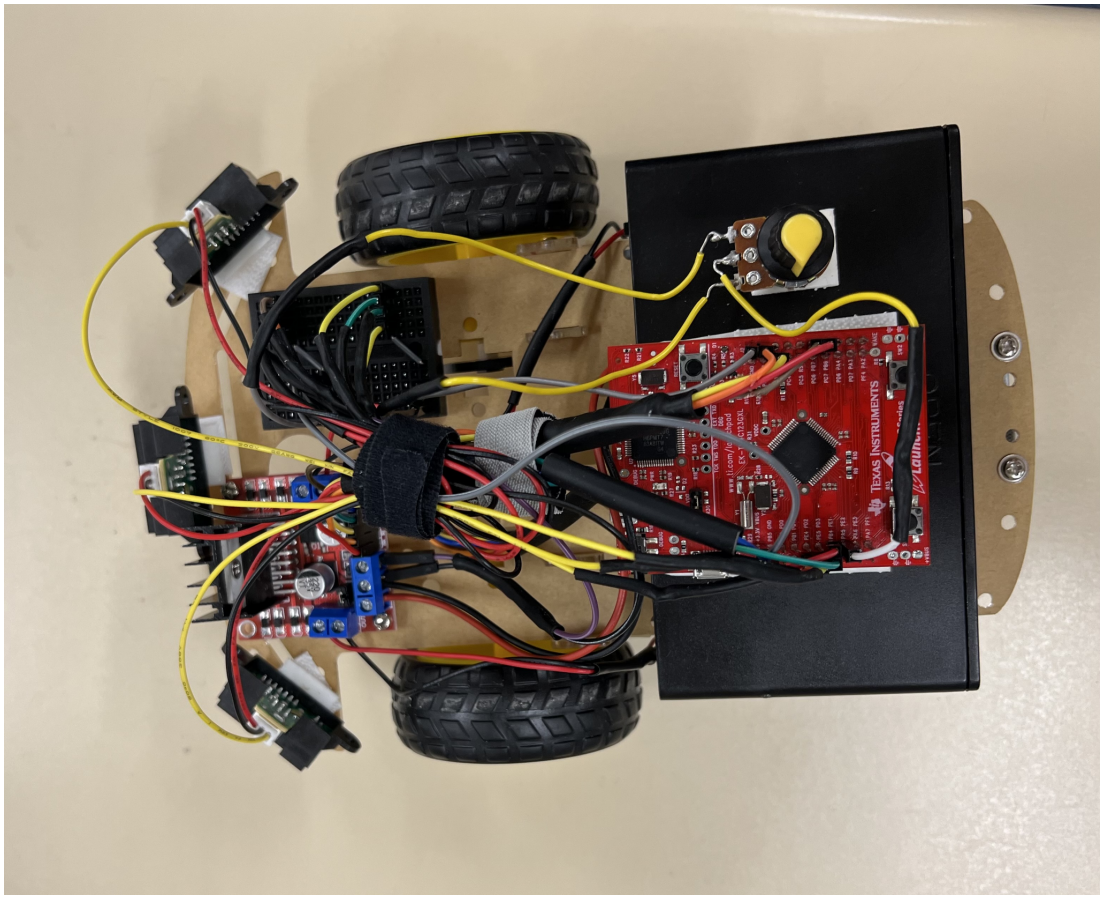
1. Middle Pass 1: https://youtu.be/4YNZLcFtx2g
2. Middle Pass 2: https://youtu.be/WkhjoEZsmlE
3. Right Pass 1: https://youtu.be/knWAwHEaixk
4. Right Pass 2: https://youtu.be/9Br__QJQqjw
5. Left Pass 1: https://youtu.be/qWEK-pSVWMU
6. Left Pass 2: https://youtu.be/xAW48l5V_f4

**Theory**

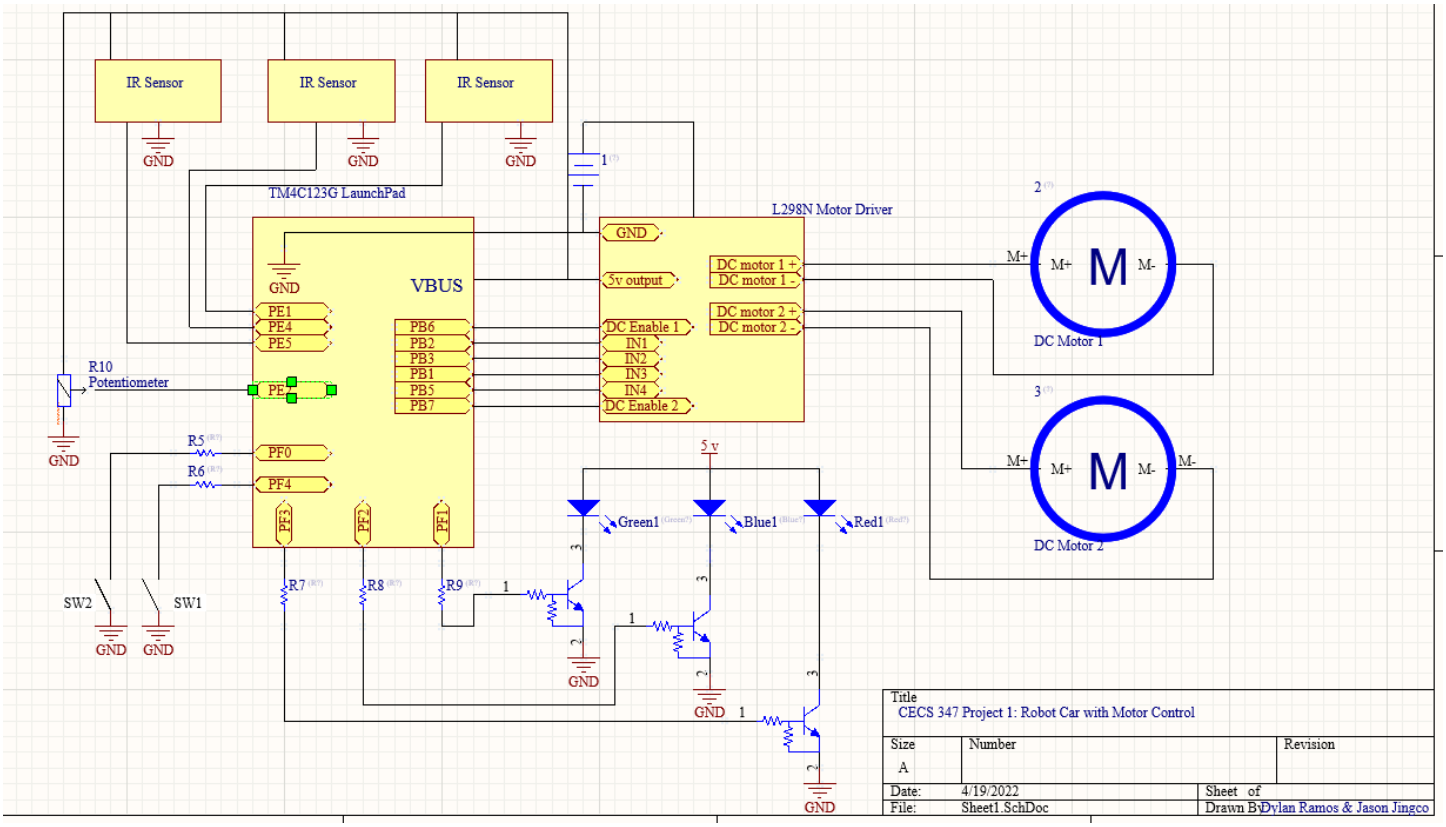
As mentioned in the introduction, we used several concepts that were covered from 346 and 347 lectures.The first concept(Interrupts) allowed us to detect both Switches on the falling edge to determine the speed flag and wheel direction. DC motors were used to control the wheels of the car controlled by an L298N H-bridge. This component uses an internal H-bridge to control the direction of

the wheels by providing a load in either direction. Since the L298N contains an onboard 5v regulator a power supply of up to 12v was able to be used to power the TM4C123G board. Three IR proximity sensors are used to compare the distance of the car to the track. The IR sensors take advantage of ADC concepts by measuring the distance taking in analog input and converting it to a digital output. Each sensor has a voltage range of 0 - 3.3v with the ADCvalue ranging from 0 to 4095. Similarly, the 10k ohm potentiometer that controls the speed of the car outputs an ADC value.

**Hardware**



**Schematic:**




**Software**

In order in implementing project 2, we used 3 C modules that all worked together to allow the robot car to navigate through a prebuilt track. The WallFollowerStarter module contains most of the functions(including the main function) that initializes the motor(both wheels), LED's, Car direction, Systick, and GPIO port F. It also includes the steering function that contains the logic for how the robot car wheels react if one of the 3 sensors is close to the wall. Another added ADC sensor was the Potentiometer to allow the change of wheel speed. The ADCMultiSamples module contains the initialization for the ADC inputs PE1(ain2), PE2(ain1), PE4(ain9), and PE5(ain8). It is using the Sample Sequencer 2 since it allows 4 inputs to be utilized. The PLL module contains the initialization for the Phase Lock Loop which allows us to set the frequency of the processor to 50MHz.

WallFollowerStarter.c:

The WallFollowerStarter.c module contains most of all the functions in order for the robot car to move based on the prebuilt track.

```
40
41  #include "ADCMultiSamples.h"
42  #include "PLL.h"
43  #include "tm4c123gh6pm.h"
44  #include "stdint.h"
45
46  // basic functions defined at end of startup.s
47  void DisableInterrupts(void); // Disable interrupts
48  void EnableInterrupts(void);  // Enable interrupts
49  void WaitForInterrupt(void);  // low power mode
50
51  uint8_t sample = 0, enableMotor = 0, speed_mode = 0;
52
53  #define LED (*((volatile unsigned long *)0x40025038))  // use onboard three LEDs: PF321
54  // You use datasheet to calculate the following ADC values
55  // then test your sensors to adjust the values
56  #define CRASH             IR15CM// if there is less than this distance ahead of the robot, it will immediately stop
57  #define IR15CM            2233  // ADC output for 15cm:1.8v -> (1.8/3.3)*4095=2233
58  #define IR20CM            1724  // ADC output for 20cm:1.39v -> (1.39/3.3)*4095=1724
59  #define IR30CM            1116  // ADC output for 30cm:0.9v -> (0.9/3.3)*4095=1116
60  #define IR40CM            918   // ADC output for 40cm:0.74v -> (0.74/3.3)*4095=918
61  #define IR80CM            496   // ADC output for 80cm:0.4v -> (0.4/3.3)*4095=496
62  #define IR60CM            707
63                                  // with equal power to both motors (LeftH == RightH), the robot still may not drive straight
64                                  // due to mechanical differences in the motors, so bias the left wheel faster or slower than
65                                  // the constant right wheel
66  #define LEFTMINPCT        30    // minimum percent duty cycle of left wheel (10 to 90)
67  #define LEFTMAXPCT        50    // maximum percent duty cycle of left wheel (10 to 90)
68  #define RIGHTCONSTPCT     40    // constant percent duty cycle of right wheel (10 to 90)
69  #define period            50000
70  #define FIFTY_PERCENT     0.5
71  #define THIRTY_PERCENT    0.3
72  #define TWENTY_PERCENT    0.2
73  #define FIFTN_PERCENT     0.15
74  #define TEN_PERCENT       0.1
75  #define RELOAD 799999
```

```
76  #define NVIC_EN0_PORTF  0x40000000     // (h) enable interrupt 30 in NVIC
77  #define WHEEL_DIR (*((volatile unsigned long *)0x400050F0)) // PB5432 are the four direction pins for L298
78  #define Dark      0x00
79  #define Red       0x02
80  #define Blue      0x04
81  #define Green     0x08
82  #define Yellow    0x0A
83  #define Cran      0x0C
84  #define White     0x0E
85  #define Purple    0x06
86  // Constant definitions based on the following hardware interface:
87  // PB5432 are used for direction control on L298.
88  // Motor 1 is connected to the left wheel, Motor 2 is connected to the right wheel.
89  #define FORWARD 0x28
90  #define BACKWARD 0x14
91  #define LEFTPIVOT 0x18
92  #define RIGHTPIVOT 0x24
93
94  void System_Init(void);
95  void LEDSW_Init(void);
96  void Motor_Init(void);
97  void SysTick_Init(void);
98  void steering(uint16_t ahead_dist,uint16_t right_dist, uint16_t left_dist);
99  void ReadADCFIRFilter(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8);
100 void ReadADCIIRFilter(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8);
101 uint16_t median(uint16_t u1, uint16_t u2, uint16_t u3);
102 void ReadADCMedianFilter(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8);// This function samples AIN1 (PE2), AIN2 (PE1), AIN9 (PE4), AIN8 (PE5)
103 void PWM_PB76_Duty(unsigned long duty_L, unsigned long duty_R);
104 void Car_Dir_Init(void);
105 void Delay(void);
106 void GPIO_PORTF_Init(void);
107
108 unsigned char eq_calcution(unsigned int ADC_Value);
109 unsigned char tb_estimation(unsigned int ADC_Value);
110 int p_adcvalue = 0;
111 double p_speed = 0;
```

```
146
147     GPIO_PORTF_AMSEL_R &= ~0x0E;       // 3) disable analog function
148     GPIO_PORTF_PCTL_R &= ~0x0000FFF0; // 4) GPIO clear bit PCTL
149     GPIO_PORTF_DIR_R |= 0x0E;          // 6) PF1-PF3 output
150     GPIO_PORTF_AFSEL_R &= ~0x0E;       // 7) no alternate function
151     GPIO_PORTF_DEN_R |= 0x0E;          // 8) enable digital pins PF3-PF1
152     LED = Dark;                        // Turn off all LEDs.
153   }
154
155 void Motor_Init(void){
156     if ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOB)==0) {
157       SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB; // Activate B clocks
158       while ((SYSCTL_RCGC2_R&SYSCTL_RCGC2_GPIOB)==0){};
159     }
160
161
162     LeftH = (LEFTMAXPCT + LEFTMINPCT)*400;
163     LeftL = 80000 - LeftH;          // value modified my controller
164     RightH = RIGHTCONSTPCT*800;     // constant
165     RightL = 80000 - RightH;
166
167
168     GPIO_PORTB_AFSEL_R |= 0xC0; // enable alt funct: PB76 for PWM
169     GPIO_PORTB_PCTL_R &= ~0xFF000000; // PWM to be used
170     GPIO_PORTB_PCTL_R |= 0x44000000; // PWM to be used
171     GPIO_PORTB_DEN_R |= 0xC0; // enable digital I/O
172
173     // Initializes PWM settings
174     SYSCTL_RCGCPWM_R |= 0x01; // activate PWM0
175     SYSCTL_RCC_R &= ~0x001E0000; // Clear any previous PWM divider values
176
```

```
215 void steering(uint16_t ahead_dist,uint16_t right_dist, uint16_t left_dist){
216     // Suggest the following simple control as starting point:
217     // 1. If any one of the senors see obstacle <20cm, stop
218     // 2. If all sensors detect no obstacle within 35cm, stop
219     // 3. If left sees obstacle within 30cm, turn right
220     // 4. If right sees obstacle within 30cm, turn left
221     // 5. If both sensors see no obstacle within 30cm, go straight
222     if(speed_mode) {
223       uint16_t ain2newest;
224       uint16_t ain9newest;
225       uint16_t ain8newest;
226       uint16_t ain1newest;
227       uint16_t *ain1;
228       static uint16_t ain1oldest=0, ain1middle=0;
229       ADC_In298(&ain2newest, &ain9newest, &ain8newest, &ain1newest);
230       ain1middle = ain1newest; ain1middle = ain1newest; ain1middle = ain1newest;
231       *ain1 = median(ain1newest, ain1middle, ain1oldest);
232       p_adcvalue = *ain1;
233
234       if(p_adcvalue > 0 && p_adcvalue <= 360) p_speed = 0.05;
235       else if(p_adcvalue > 360 && p_adcvalue <= 720) p_speed = 0.15;
236       else if(p_adcvalue > 720 && p_adcvalue <= 1080) p_speed = 0.25;
237       else if(p_adcvalue > 1080 && p_adcvalue <= 1440) p_speed = 0.35;
238       else if(p_adcvalue > 1440) p_speed = 0.40;
239       else p_speed = 0;
240     }else {
241       p_adcvalue = 0;
242       p_speed = 0;
243     }
244
```

```
  if(enableMotor) {
      PWM0_ENABLE_R |= 0x00000003; // enable both wheel
      if((left_dist>IR15CM)&&(right_dist>IR15CM)&&(ahead_dist>IR15CM)){//stop moving
          PWM_PB76_Duty(0,0);
          LED=Red;
      }
      else if((left_dist>IR15CM)||(right_dist>IR15CM)){               //adjust left or right
        if(left_dist < right_dist){
          PWM_PB76_Duty((p_speed+FIFTN_PERCENT)*period,(p_speed+FIFTY_PERCENT)*period);
          LED=Green;
        }
        else{
          PWM_PB76_Duty((p_speed+FIFTY_PERCENT)*period,(p_speed+FIFTN_PERCENT)*period);
          LED=Blue;
        }
      }
      else if((left_dist<IR60CM)&&(right_dist<IR60CM)&&(ahead_dist<IR60CM)){  //open area
        PWM_PB76_Duty(0,0);
        LED=Purple;
      }

      else if((left_dist>IR20CM)){  //right moving
        PWM_PB76_Duty((p_speed+FIFTY_PERCENT)*period,(p_speed+FIFTN_PERCENT)*period);

      }
      else if((right_dist>IR20CM)){ // moving

        PWM_PB76_Duty((p_speed+FIFTN_PERCENT)*period,(p_speed+FIFTY_PERCENT)*period);

      }
      else if(left_dist<right_dist){
        LED=Dark;
        if((right_dist-left_dist) > IR20CM){
          PWM_PB76_Duty((p_speed+THIRTY_PERCENT)*period,(p_speed+THIRTY_PERCENT)*period);
        }
        else
          PWM_PB76_Duty((p_speed+FIFTN_PERCENT)*period,(p_speed+THIRTY_PERCENT)*period);
      }
      else if(left_dist>right_dist){
        LED=Dark;
        if((left_dist-right_dist) > IR20CM){
          PWM_PB76_Duty((p_speed+THIRTY_PERCENT)*period,(p_speed+THIRTY_PERCENT)*period);
        }
        else
          PWM_PB76_Duty((p_speed+THIRTY_PERCENT)*period,(p_speed+FIFTN_PERCENT)*period);
      }
      else {//if((right_dist>IR60CM)&&(left_dist>IR60CM)){
        LED=Dark;
        PWM_PB76_Duty((p_speed+THIRTY_PERCENT)*period,(p_speed+THIRTY_PERCENT)*period);
      }
  }else {
    LED = 0;
    PWM0_ENABLE_R &= ~0x00000003; // disable both wheels
  }

  //  }
  // Feel free to add more controlls to fine tune your robot car.
  // Make sure to take care of both wheel movements and LED display here.

}
void SysTick_Handler(void){
  sample = 1;
}
```

```
// returns the results in the corresponding variables.  Some
// kind of filtering is required because the IR distance sensors
// output occasional erroneous spikes.  This is an FIR filter:
// y(n) = (x(n) + x(n-1))/2
// Assumes: ADC initialized by previously calling ADC_Init298()
void ReadADCFIRFilter(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8){
  static uint16_t ain2previous=0; // after the first call, the value changed to 12
  static uint16_t ain9previous=0;
  static uint16_t ain8previous=0;
  // save some memory; these do not need to be 'static'
  //            x(n)
  uint16_t ain2newest;
  uint16_t ain9newest;
  uint16_t ain8newest;
  uint16_t ain1newest;
  ADC_In298(&ain2newest, &ain9newest, &ain8newest, &ain1newest); // sample AIN1 (PE2), AIN2(PE1), AIN9 (PE4), AIN8 (PE5)
  *ain2 = (ain2newest + ain2previous)/2;
  *ain9 = (ain9newest + ain9previous)/2;
  *ain8 = (ain8newest + ain8previous)/2;
  ain2previous = ain2newest; ain9previous = ain9newest; ain8previous = ain8newest;
}

// This function samples AIN2 (PE1), AIN9 (PE4), AIN8 (PE5) and
// returns the results in the corresponding variables.  Some
// kind of filtering is required because the IR distance sensors
// output occasional erroneous spikes.  This is an IIR filter:
// y(n) = (x(n) + y(n-1))/2
// Assumes: ADC initialized by previously calling ADC_Init298()
void ReadADCIIRFilter(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8){
  //                   y(n-1)
  static uint16_t filter2previous=0;
  static uint16_t filter9previous=0;
  static uint16_t filter8previous=0;
  // save some memory; these do not need to be 'static'
  //            x(n)
  uint16_t ain2newest;
  uint16_t ain9newest;
  uint16_t ain8newest;
  uint16_t ain1newest;
  ADC_In298(&ain2newest, &ain9newest, &ain8newest, &ain1newest); // sample AIN1 (PE2), AIN2(PE1), AIN9 (PE4), AIN8 (PE5)
  *ain2 = filter2previous = (ain2newest + filter2previous)/2;
  *ain9 = filter9previous = (ain9newest + filter9previous)/2;
  *ain8 = filter8previous = (ain8newest + filter8previous)/2;
}

// Median function from EE345M Lab 7 2011; Program 5.1 from Volume 3
// helper function for ReadADCMedianFilter() but works for general use
uint16_t median(uint16_t u1, uint16_t u2, uint16_t u3){
uint16_t result;
  if(u1>u2)
    if(u2>u3)   result=u2;     // u1>u2,u2>u3       u1>u2>u3
      else
        if(u1>u3) result=u3;   // u1>u2,u3>u2,u1>u3 u1>u3>u2
        else      result=u1;   // u1>u2,u3>u2,u3>u1 u3>u1>u2
  else
    if(u3>u2)   result=u2;     // u2>u1,u3>u2       u3>u2>u1
      else
        if(u1>u3) result=u1;   // u2>u1,u2>u3,u1>u3 u2>u1>u3
        else      result=u3;   // u2>u1,u2>u3,u3>u1 u2>u3>u1
  return(result);
}

// This function samples AIN2 (PE1), AIN9 (PE4), AIN8 (PE5) and
// returns the results in the corresponding variables.  Some
// kind of filtering is required because the IR distance sensors
// output occasional erroneous spikes.  This is a median filter:
// y(n) = median(x(n), x(n-1), x(n-2))
// Assumes: ADC initialized by previously calling ADC_Init298()
void ReadADCMedianFilter(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8){
  //                      x(n-2)       x(n-1)
  static uint16_t ain2oldest=0, ain2middle=0;
  static uint16_t ain9oldest=0, ain9middle=0;
  static uint16_t ain8oldest=0, ain8middle=0;
```

```
  // save some memory; these do not need to be 'static'
  //            x(n)
  uint16_t ain2newest;
  uint16_t ain9newest;
  uint16_t ain8newest;
  uint16_t ain1newest;
  ADC_In298(&ain2newest, &ain9newest, &ain8newest, &ain1newest); // sample AIN2(PE1), AIN9 (PE4), AIN8 (PE5)
  *ain2 = median(ain2newest, ain2middle, ain2oldest);
  *ain9 = median(ain9newest, ain9middle, ain9oldest);
  *ain8 = median(ain8newest, ain8middle, ain8oldest);
  ain2oldest = ain2middle; ain9oldest = ain9middle; ain8oldest = ain8middle;
  ain2middle = ain2newest; ain9middle = ain9newest; ain8middle = ain8newest;

}

void GPIO_PORTF_Init(void){
  unsigned long volatile delay;
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOF; // activate clock for port F
  delay = SYSCTL_RCGC2_R;
  GPIO_PORTF_LOCK_R = 0x4C4F434B;       // unlock GPIO Port F
  GPIO_PORTF_CR_R |= 0x11;              // allow changes for PF4,PF0 take effect
  GPIO_PORTF_AMSEL_R &= ~0x11;          // disable analog functionality on PF4,PF0
  GPIO_PORTF_PCTL_R &= ~0x000F000F;     // configure PF4, PF0 as GPIO
  GPIO_PORTF_DIR_R &= ~0x11;            // make PF4,0 in (built-in button)
  GPIO_PORTF_DEN_R |= 0x11;             // enable digital I/O on PF4, PF0
  GPIO_PORTF_AFSEL_R &= ~0x11;          // disable alt funct on PF4, PF0
  GPIO_PORTF_PUR_R |= 0x11;             // enable weak pull-up on PF4,PF0
  GPIO_PORTF_IS_R &= ~0x11;             // PF4,PF0 is edge-sensitive
  GPIO_PORTF_IBE_R &= ~0x11;            // PF4,PF0 is not both edges
  GPIO_PORTF_IEV_R &= ~0x11;            // PF4,PF0 falling edge event
  GPIO_PORTF_ICR_R = 0x11;              // clear flags 4,0
  GPIO_PORTF_IM_R |= 0x11;              // arm interrupt on PF4,PF0
  NVIC_PRI7_R = (NVIC_PRI7_R&0xFF1FFFFF)|0x00400000; // bits:23-21 for PORTF, set priority to 2
  NVIC_EN0_R |= NVIC_EN0_PORTF;         // enable interrupt 30 in NVIC
}
```

```

// PORTF ISR:
// Change delivered power based on switch press:
void GPIOPortF_Handler(void){ // called on touch of either SW1 or SW2
  if(GPIO_PORTF_RIS_R&0x01) {  // SW2 touched - activates wheel speed mode
    Delay();
    GPIO_PORTF_ICR_R = 0x01;  // acknowledge flag0
    speed_mode ^= 1 + 0;
    LED = White;
    Delay();
  }
  if(GPIO_PORTF_RIS_R&0x10){  // SW1 touched - enables or disables wheel movement.
    Delay();
    GPIO_PORTF_ICR_R = 0x10;  // acknowledge flag4
    enableMotor ^= 1 + 0; // enable or disable wheels
    LED =Yellow;
    Delay();
  }
}

void Delay(void) {
  unsigned long volatile time;
  time = 727240*70/91;  // 0.03sec
  while(time){
    time--;
  }
}
```

ADCMultiSamples.c:

The ADCMultiSamples contains the initialization for the ADC inputs PE1, PE2, PE4, and PE5 to read the ADC value from the sensors(3 proximity sensors and 1 potentiometer).

```
61
62  void ADC_Init298(void){
63    volatile unsigned long delay;
64  //  SYSCTL_RCGC0_R |= 0x00010000; // 1) activate ADC0 (legacy code)
65    SYSCTL_RCGCADC_R |= 0x00000001; // 1) activate ADC0
66    SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R4; // 1) activate clock for Port E
67    delay = SYSCTL_RCGCGPIO_R;      // 2) allow time for clock to stabilize
68    delay = SYSCTL_RCGCGPIO_R;
69    GPIO_PORTE_DIR_R &= ~0x36;      // 3) make PE1, PE2, PE4, and PE5 input
70    GPIO_PORTE_AFSEL_R |= 0x36;     // 4) enable alternate function on PE1, PE2, PE4, and PE5
71    GPIO_PORTE_DEN_R &= ~0x36;      // 5) disable digital I/O on PE1, PE2, PE4, and PE5
72                                    // 5a) configure PE4 as ?? (skip this line because PCTL is for digital only)
73    GPIO_PORTE_PCTL_R = GPIO_PORTE_PCTL_R&0xFF00F00F;
74    GPIO_PORTE_AMSEL_R |= 0x36;     // 6) enable analog functionality on PE1, PE2, PE4, and PE5
75    ADC0_PC_R &= ~0xF;              // 8) clear max sample rate field
76    ADC0_PC_R |= 0x1;               //    configure for 125K samples/sec
77    ADC0_SSPRI_R = 0x3210;          // 9) Sequencer 3 is lowest priority
78    ADC0_ACTSS_R &= ~0x0004;        // 10) disable sample sequencer 2
79    ADC0_EMUX_R &= ~0x0F00;         // 11) seq2 is software trigger
80    ADC0_SSMUX2_R = 0x1892;         // 12) set channels for SS2
81    ADC0_SSCTL2_R = 0x0600;         // 13) no D0 END0 IE0 TS0 D1 END1 IE1 TS1 D2 TS2, yes END2 IE2
82    ADC0_IM_R &= ~0x0004;           // 14) disable SS2 interrupts
83    ADC0_ACTSS_R |= 0x0004;         // 15) enable sample sequencer 2
84  }
85
```

```
86  //------------ADC_In298------------
87  // Busy-wait Analog to digital conversion
88  // Input: none
89  // Output: three 12-bit result of ADC conversions
90  // Samples AIN8, AIN9, AIN2, and AIN1
91  // 125k max sampling
92  // software trigger, busy-wait sampling
93  // data returned by reference
94  // ain2 (PE1) 0 to 4095
95  // ain9 (PE4) 0 to 4095
96  // ain8 (PE5) 0 to 4095
97  // ain1 (PE2) 0 to 1450
98  void ADC_In298(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8, uint16_t *ain1){
99    ADC0_PSSI_R = 0x0004;             // 1) initiate SS2
100   while((ADC0_RIS_R&0x04)==0){};    // 2) wait for conversion done
101   *ain2 = ADC0_SSFIFO2_R&0xFFF;     // 3A) read first result
102   *ain9 = ADC0_SSFIFO2_R&0xFFF;     // 3B) read second result
103   *ain8 = ADC0_SSFIFO2_R&0xFFF;     // 3C) read third result
104   *ain1 = ADC0_SSFIFO2_R&0xFFF;     // P-Monitor
105   ADC0_ISC_R = 0x0004;              // 4) acknowledge completion
106 }
107
```

ADCMultiSamples.h:

The ADCMultiSamples.h contains the function calls for ADC init functions to be called in the WallFollowerStarter module.

```
47
48  #include "stdint.h"
49
50  // Initializes AIN2, AIN9, AIN8, and AIN1 sampling
51  // 125k max sampling
52  // SS2 triggering event: software trigger, busy-wait sampling
53  // SS2 1st sample source: AIN2 (PE1)
54  // SS2 2nd sample source: AIN9 (PE4)
55  // SS2 3rd sample source: AIN8 (PE5)
56  // SS2 4th sample source: AIN1 (PE2)
57  // SS2 interrupts: enabled after 3rd sample but not promoted to controller
58  void ADC_Init298(void);
59
60  //------------ADC_In298------------
61  // Busy-wait Analog to digital conversion
62  // Input: none
63  // Output: three 12-bit result of ADC conversions
64  // Samples AIN8, AIN9, AIN2, and AIN1
65  // 125k max sampling
66  // software trigger, busy-wait samplingS
67  // data returned by reference
68  // ain2 (PE1) 0 to 4095
69  // ain9 (PE4) 0 to 4095
70  // ain8 (PE5) 0 to 4095
71  // ain1 (PE2) 0 to 1450
72  void ADC_In298(uint16_t *ain2, uint16_t *ain9, uint16_t *ain8, uint16_t *ain1);
73
```

PLL.c:

The PLL.c module allows us to use the external oscillator for a more accurate clock as well as allowing us to change to a higher clock speed.  Currently we have this set to 50MHz for project 2.

```
#include "PLL.h"

// The #define statement SYSDIV2 in PLL.h
// initializes the PLL to the desired frequency.

// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz
// see the table at the end of this file

#define SYSCTL_RIS_R            (*((volatile unsigned long *)0x400FE050))
#define SYSCTL_RIS_PLLLRIS      0x00000040  // PLL Lock Raw Interrupt Status
#define SYSCTL_RCC_R            (*((volatile unsigned long *)0x400FE060))
#define SYSCTL_RCC_XTAL_M       0x000007C0  // Crystal Value
#define SYSCTL_RCC_XTAL_6MHZ    0x000002C0  // 6 MHz Crystal
#define SYSCTL_RCC_XTAL_8MHZ    0x00000380  // 8 MHz Crystal
#define SYSCTL_RCC_XTAL_16MHZ   0x00000540  // 16 MHz Crystal
#define SYSCTL_RCC2_R           (*((volatile unsigned long *)0x400FE070))
#define SYSCTL_RCC2_USERCC2     0x80000000  // Use RCC2
#define SYSCTL_RCC2_DIV400      0x40000000  // Divide PLL as 400 MHz vs. 200
                                            // MHz
#define SYSCTL_RCC2_SYSDIV2_M   0x1F800000  // System Clock Divisor 2
#define SYSCTL_RCC2_SYSDIV2LSB  0x00400000  // Additional LSB for SYSDIV2
#define SYSCTL_RCC2_PWRDN2      0x00002000  // Power-Down PLL 2
#define SYSCTL_RCC2_BYPASS2     0x00000800  // PLL Bypass 2
#define SYSCTL_RCC2_OSCSRC2_M   0x00000070  // Oscillator Source 2
#define SYSCTL_RCC2_OSCSRC2_MO  0x00000000  // MOSC

// configure the system to get its clock from the PLL
void PLL_Init(void){
  // 0) configure the system to use RCC2 for advanced features
  //    such as 400 MHz PLL and non-integer System Clock Divisor
  SYSCTL_RCC2_R |= SYSCTL_RCC2_USERCC2;
  // 1) bypass PLL while initializing
  SYSCTL_RCC2_R |= SYSCTL_RCC2_BYPASS2;
  // 2) select the crystal value and oscillator source
  SYSCTL_RCC_R &= ~SYSCTL_RCC_XTAL_M;   // clear XTAL field
  SYSCTL_RCC_R += SYSCTL_RCC_XTAL_16MHZ;// configure for 16 MHz crystal
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_OSCSRC2_M;// clear oscillator source field
  SYSCTL_RCC2_R += SYSCTL_RCC2_OSCSRC2_MO;// configure for main oscillator source
  // 3) activate PLL by clearing PWRDN
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_PWRDN2;
  // 4) set the desired system divider and the system divider least significant bit
  SYSCTL_RCC2_R |= SYSCTL_RCC2_DIV400;  // use 400 MHz PLL
  SYSCTL_RCC2_R = (SYSCTL_RCC2_R&~ 0x1FC00000)  // clear system clock divider
                  + (SYSDIV2<<22);      // configure for 80 MHz clock
  // 5) wait for the PLL to lock by polling PLLLRIS
  while((SYSCTL_RIS_R&SYSCTL_RIS_PLLLRIS)==0){};
  // 6) enable use of PLL by clearing BYPASS
  SYSCTL_RCC2_R &= ~SYSCTL_RCC2_BYPASS2;
}
```

PLL.h

The PLL.h module contains the SYSDIV2 value that will be utilized in the PLL_Init function to calculate the frequency. Currently we have it set to 7 to allow the clock frequency to be at 50MHz.

```
// The #define statement SYSDIV2 initializes
// the PLL to the desired frequency.
#define SYSDIV2 7
// bus frequency is 400MHz/(SYSDIV2+1) = 400MHz/(7+1) = 50 MHz

// configure the system to get its clock from the PLL
void PLL_Init(void);


/*
SYSDIV2  Divisor  Clock (MHz)
 0        1       reserved
 1        2       reserved
 2        3       reserved
 3        4       reserved
 4        5       80.000
 5        6       66.667
 6        7       reserved
 7        8       50.000
 8        9       44.444
 9        10      40.000
 10       11      36.364
 11       12      33.333
 12       13      30.769
 13       14      28.571
 14       15      26.667
 15       16      25.000
 16       17      23.529
 17       18      22.222
 18       19      21.053
 19       20      20.000
 20       21      19.048
```

**Conclusion**

This project was extremely interesting project because it was quite complex, but enjoyable to create. However, we did face certain challenges. The first challenge was balancing the clock speed of the system on the TM4C123 Launchpad because the if the clock speed is too fast the IR sensors struggled to sample ADC values. The solution we decided on was to decrease the clock speed of the entire system which allowed for a beneficial balance for the motors and IR sensors. The last challenge was testing the car movement funcitonality. Testing the cars movement and turns was difficult because it required supplies and resources that we did not have. So the only time we could test was during lab time. Although we encountered these challenges, it helped us understand the logic from both a software and hardware perspective solution. Also, comparing each others project in a friendly, but competitive race was extremely fun. Overall, this lab was a great learning experience because it allowed us to combine everything we learned thus far into a single project..