EE445M Lab 7 Report

Yen-Kai Huang, Siavash Zanganeh Kamali, Chen Cui, Miao Qi, Yan Zhang

May 5, 2014

1 Objective

This is the final robotic project. We build a robot and compete in the final race with other robots.

2 Hardware Design

(a) Final mechanical drawing of the robot See Figure-1

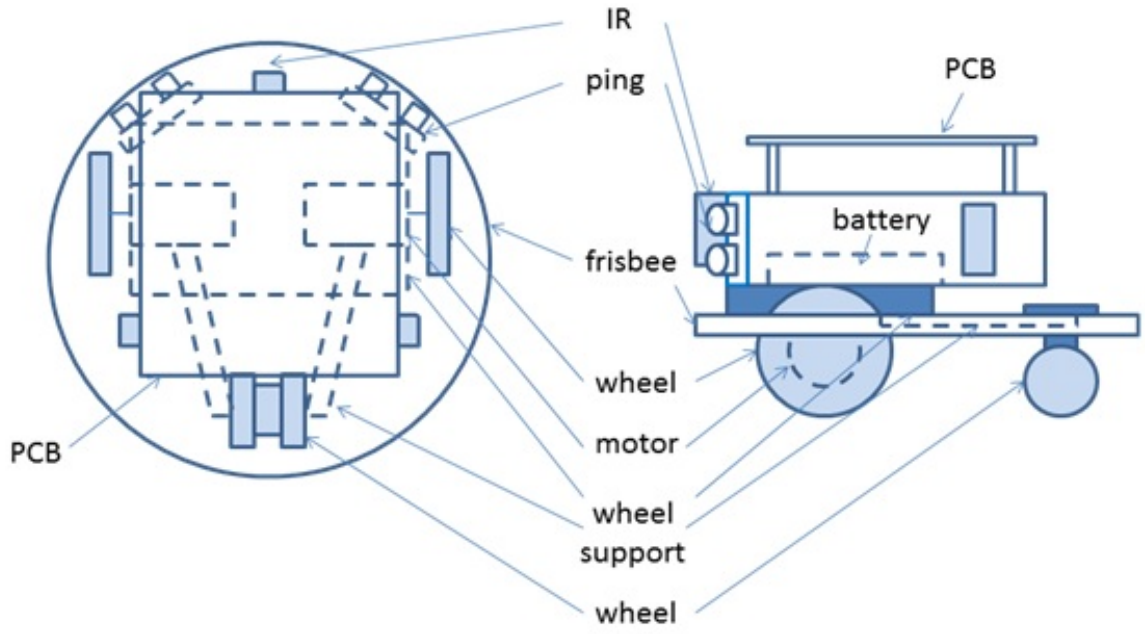

Figure 1: Mechanical Drawing

(b) Final electrical circuit diagram for the motor interfaces See Figure-2

(c) Final power supply circuitry See Figure-3

(d) Final electrical circuit diagram for the sensor interfaces
See Figure-4 for Ping circuits and Figure-5 for IR sensor circuits.

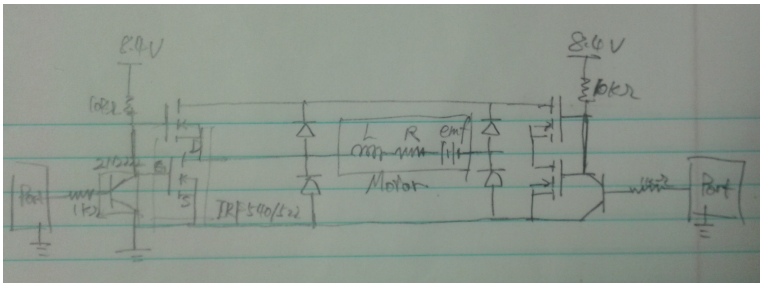

Figure 2: Motor Circuit

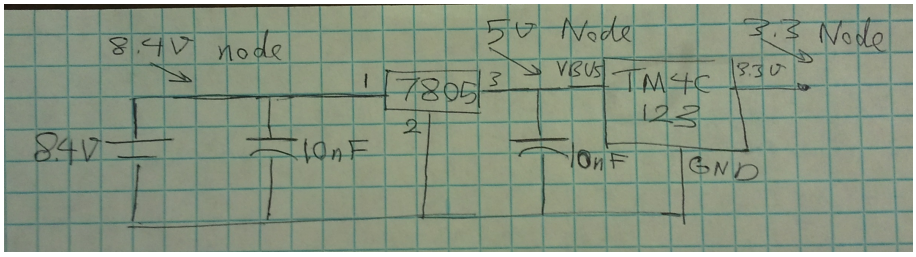

Figure 3: Power Circuit

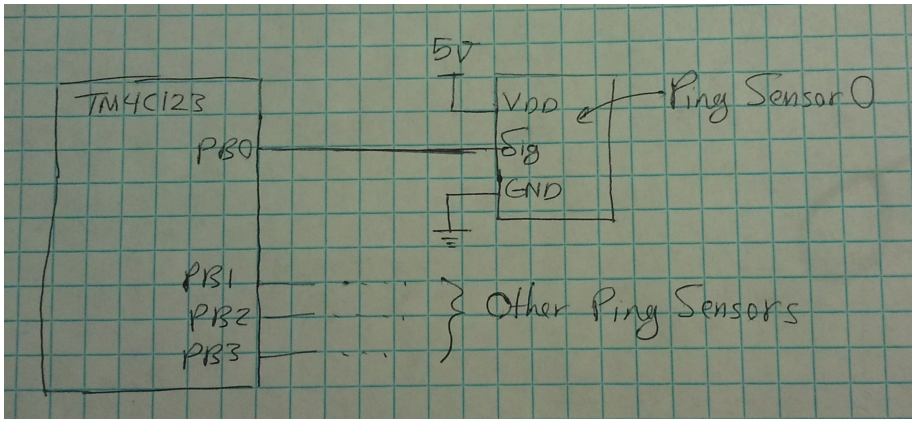

Figure 4: Ping Circuit

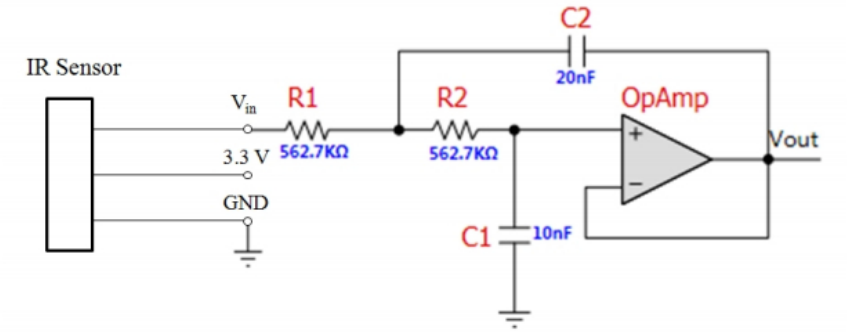

Figure 5: IR Circuit

3 Software Design

(a) Low-level device drivers for the motor interfaces (header and code files)
See below

```
// motor.h
// Runs on TM4C123
// Use PWM0/PB6 to generate pulse-width modulated outputs.
// Daniel Valvano
// September 3, 2013

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to ARM Cortex M
   Microcontrollers",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2013
  Program 6.7, section 6.3.2

 Copyright 2013 by Jonathan W. Valvano, valvano@mail.utexas.edu
    You may use, edit, run or distribute this file
    as long as the above copyright notice remains
 THIS SOFTWARE IS PROVIDED "AS IS".  NO WARRANTIES, WHETHER EXPRESS,
   IMPLIED
 OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF
 MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE APPLY TO THIS
   SOFTWARE.
 VALVANO SHALL NOT, IN ANY CIRCUMSTANCES, BE LIABLE FOR SPECIAL,
   INCIDENTAL,
 OR CONSEQUENTIAL DAMAGES, FOR ANY REASON WHATSOEVER.
 For more information about my classes, my research, and my books, see
 http://users.ece.utexas.edu/~valvano/
 */

#ifndef __MOTOR_H__
#define __MOTOR_H__

#define MOTOR_CW  0
#define MOTOR_CCW 1

#define Carpet_Floor  1
#define Smooth_Floor  2
#define Floor         Carpet_Floor

#if Floor == Carpet_Floor
#define MOTOR_DIFF   750
#else
#define MOTOR_DIFF   200
#endif

// period is 16-bit number of PWM clock cycles in one period (3<=period
   )
// duty is number of PWM clock cycles output is high  (2<=duty<=period
   -1)
// PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
//                = BusClock/2
//                = 50 MHz/2 = 25 MHz (in this example)
void Motor_Init(unsigned short duty);
```

```
46
47 // change duty cycle
48 // duty is number of PWM clock cycles output is high  (2<=duty<=period)
49 void Motor_MotionUpdate(long duty0, long duty1);
50 // Completely stop both motors
51 void Motor_Stop(void);
52
53 #endif // __MOTOR_H__
```

code/motor.h

```
1  // motor.c
2  #include "motor.h"
3  #include <tm4c123gh6pm.h>
4
5  #define PB7      (*((volatile unsigned long *)0x40005200))
6  #define PB5      (*((volatile unsigned long *)0x40005080))
7
8  #define PERIOD       25000
9
10 // period is 16-bit number of PWM clock cycles in one period (3<=period
       )
11 // duty is number of PWM clock cycles output is high  (2<=duty<=period
       -1)
12 // PWM clock rate = processor clock rate/SYSCTL_RCC_PWMDIV
13 //                = BusClock/2
14 //                = 50 MHz/2 = 25 MHz (in this example)
15 void Motor_Init(unsigned short duty){   volatile unsigned long delay;
16
17   /*************** New Style ******************/
18   SYSCTL_RCGCPWM_R |= SYSCTL_RCGCPWM_R0;   // 1) activate PWM0
19     SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1; // 2) activate port B
20     delay = SYSCTL_RCGCGPIO_R;               // allow time to finish
       activating
21   /*******************************************/
22
23   GPIO_PORTB_AFSEL_R |= 0x50;           // enable alt funct on PB4,6
24   GPIO_PORTB_AFSEL_R &= ~0xA0;      // disable alt funct on PB5,7
25   GPIO_PORTB_DIR_R  |= 0xA0;      // set PB5,7 output
26   GPIO_PORTB_PCTL_R = (GPIO_PORTB_PCTL_R &~0x0F0F0000) | 0x04040000;
       // configure PB4,6 as PWM0
27   GPIO_PORTB_AMSEL_R &= ~0xF0;          // disable analog functionality
       on PB4,5,6,7
28   GPIO_PORTB_DEN_R |= 0xF0;             // enable digital I/O on PB4
       ,5,6,7
29
30   PB5 = PB7 = MOTOR_CW;
31
32   SYSCTL_RCC_R |= SYSCTL_RCC_USEPWMDIV; // 3) use PWM divider
33   SYSCTL_RCC_R &= ~SYSCTL_RCC_PWMDIV_M; //    clear PWM divider field
34   SYSCTL_RCC_R |= SYSCTL_RCC_PWMDIV_2;  //    configure for /2 divider
35
36   PWM0_0_CTL_R = PWM0_1_CTL_R = 0;      // 4) re-loading mode
37   PWM0_0_GENA_R = (PWM_0_GENA_ACTCMPAD_ONE|PWM_0_GENA_ACTLOAD_ZERO);
38   PWM0_1_GENA_R = (PWM_1_GENA_ACTCMPAD_ONE|PWM_1_GENA_ACTLOAD_ZERO);
39
```

```
40    PWM0_0_LOAD_R = PWM0_1_LOAD_R = PERIOD - 1;           // 15) cycles
       needed to count down to 0
41    PWM0_0_CMPA_R = PWM0_1_CMPA_R = duty - 1;             // 6) count
       value when output rises
42
43    PWM0_0_CTL_R |= PWM_0_CTL_ENABLE;     // 7) start PWM0
44    PWM0_1_CTL_R |= PWM_1_CTL_ENABLE;  //!!
45
46    PWM0_ENABLE_R |= (PWM_ENABLE_PWM0EN | PWM_ENABLE_PWM2EN);   // enable
        PWM0
47  }
48
49  // Input: [-(PERIOD-2), -2] U [2, PERIOD-2]
50  void Motor_MotionUpdate(long duty0, long duty1){
51    char dir0, dir1;
52    if (!(dir0 = (duty0>0))) {
53      duty0 = -duty0;
54    }
55    duty0 += MOTOR_DIFF;
56
57    if (!(dir1 = (duty1>0))) {
58      duty1 = -duty1;
59    }
60
61    PB7 = dir0 << 7;
62    PB5 = dir1 << 5;
63
64    if (duty0<2) duty0 = 2;
65    if (duty0>PERIOD-2) duty0 = PERIOD -2;
66    if (duty1<2) duty1 = 2;
67    if (duty1>PERIOD-2) duty1 = PERIOD -2;
68
69    PWM0_0_CMPA_R = (dir0)? (PERIOD - duty0):(duty0);
70    PWM0_1_CMPA_R = (dir1)? (PERIOD - duty1):(duty1);
71
72  }
73
74  void Motor_Stop(void) {
75    PB7 = PB5 = 0;
76    PWM0_0_CMPA_R = PWM0_1_CMPA_R = 2;
77  }
```

code/motor.c

(b) Low-level device drivers for the sensor interfaces (header and code files)

IR sensor code is basically the same as Lab. 6, but with filter removed. A calibration function is added that applies linear interpolation to find out the real distance.

```
1  // IR_sensor.h
2
3  #ifndef __IR_SENSOR_H__
4  #define __IR_SENSOR_H__
5
6  #define LAB_DEMO 7
7
8  void IR_Init(void);
```

```
9  void IR_getValues (unsigned char *buffer);
10
11 #endif // __IR_SENSOR_H__
```

code/ir_sensor.h

```
1  // IR_sensor.c
2
3  #include "ir_sensor.h"
4
5  #include "PLL.h"
6  #include "SysTick.h"
7  #include "OS.h"
8  #include "FIFO_sema4.h"
9  #include "semaphore.h"
10 #include "myadc.h"
11 #include "ST7735.h"
12 #include "interpreter.h"
13
14 #include "gpio_debug.h"
15
16 #define SAMPLING_RATE          2000 // in unit of Hz
17 #define TIMESLICE 2*TIME_1MS   // thread switch time in system time
       units
18
19 static unsigned short data[4];
20
21 /********************* Calibration detail ******************/
22 #define Calibtable_len        13
23 static const short Calitable[Calibtable_len+1] = {
24   2930, 2020, 1600, 1300, 1110, 980, 860, 780, 720, 660, 620, 560, 540,
25
26   -1, // min
27 };
28
29 // Calidiff[i] = Calitable[i] - Calitable[i+1]
30 static const short Calidiff[Calibtable_len-1] = {
31   910, 420, 300, 190, 130, 120, 80, 60, 60, 40, 60, 20,
32 };
33
34 static unsigned char calibrate(unsigned short adcval) {
35   int i;
36
37   for(i = 0; adcval <= Calitable[i] ; i++);
38   // now, C[i] < adcval <= C[i+1]
39
40   // saturation
41   if (i == 0) return 10;
42   else if(i == Calibtable_len) return 70;
43   else return (unsigned char) (10+5*(i-1) + 5*(adcval - Calitable[i])/(
       Calidiff[i-1]));
44 }
45
46
47
48 // Call back function passed to ADC
```

```
49 void  IRCallBack(unsigned short buf[]) {
50   data[0] = buf[0];
51   data[1] = buf[1];
52   data[2] = buf[2];
53   data[3] = buf[3];
54 }
55
56
57 void IR_Init(void) {
58   myADC_Collect4(SAMPLING_RATE, IRCallBack);
59 }
60
61 void IR_getValues (unsigned char *buffer) {
62   buffer[0] = calibrate(data[0]);
63   buffer[1] = calibrate(data[1]);
64   buffer[2] = calibrate(data[2]);
65   buffer[3] = calibrate(data[3]);
66 }
```

code/ir_sensor.c

The Ping))) sensor code is similar to Lab. 6 but now a median filter is added to stabilize the value. Also the number of sensor is optimized to 2.

```
1  // Ping.h
2  // Runs on LM4C123
3  // Initialize Ping interface, then generate 5us pulse about 10 times
       per second
4  // capture input pulse and record pulse width
5  // Miao Qi
6  // October 27, 2012
7
8  //initialize PB4-0
9  //PB4 set as output to send 5us pulse to all four Ping))) sensors at
       same time
10 //PB3-0 set as input to capture input from sensors
11 void Ping_Init(void);
12
13
14 // Return: the number of times this sensor has failed
15 unsigned char PingValue(unsigned char *mbox, unsigned char pingNum);
16
17 //final distance data
18 extern unsigned long Ping_Distance_Result[4];
```

code/ping.h

```
1  // Ping.c
2  // Runs on LM4C123
3  // Initialize Ping interface, then generate 5us pulse about 10 times
       per second
4  // capture input pulse and record pulse width
5  // Miao Qi
6  // October 27, 2012
7
8  // Modified
9  // Nicholas Huang
```

```
// 2014/4/19
// Use semaphore to synchronize interface

#include "inc/tm4c123gh6pm.h"
#include "OS.h"
#include "semaphore.h"

//#define Sensors           (*((volatile unsigned long *)0x4000503C))
//#define PB3_0             0x0F
#define Sensors           (*((volatile unsigned long *)0x4000500C))
#define PB3_0             0x03
#define Temperature       20
#define NVIC_EN0_INT1     2

#define TimeGap           5 // in 10 ms

#define numSensor         2

Sema4Type Sema4PingResultAvailable[numSensor], Sema4PingIdle;
long StartCritical (void);    // previous I bit, disable interrupts
void EndCritical(long sr);    // restore I bit to previous value

unsigned char PingNum=0;

static unsigned long LastStatus;
static unsigned long Starttime[numSensor];
static unsigned long Finishtime[numSensor];
static unsigned char Edge_Valid[numSensor] = {0,}; // flag

static unsigned char Distance_Result[numSensor];
static unsigned char Sensor_fail[numSensor] = {0,};

static void Ping_measure(unsigned char number);

#define APPLY_FILTER 1

#if APPLY_FILTER
/************************ Median Filter ******************/
static unsigned short median3(unsigned short *buf3){
  if(buf3[0] > buf3[1]) {
    if(buf3[0] < buf3[2]) return buf3[0]; // 2 0 1
    else return (buf3[1] > buf3[2]) ? buf3[1] : buf3[2]; // 0 1 2 or 0
   2 1
  } else { // 1 > 0
    if(buf3[0] > buf3[2]) return buf3[0]; // 1 0 2
    else return (buf3[1] > buf3[2]) ? buf3[2] : buf3[1]; // 1 2 0 or 2
   1 0
  }
}

typedef struct {
  unsigned short buf[6];
  unsigned char index;
} MedFilter;

MedFilter filter[numSensor] = {{{0}, 5},{{0}, 5}};
```

```
64
65 unsigned short MedianFilter(MedFilter *f, unsigned short n) {
66   unsigned char i = f->index;
67   if(++(f->index) == 6) f->index = 3;
68
69   f->buf[i-3] = f->buf[i] = n;
70   return median3(&f->buf[i-3+1]);
71 }
72
73 #endif
74
75 void Ping_Thread(void) {
76   while(1) {
77     Ping_measure(0);
78     Ping_measure(1);
79 //    Ping_measure(2);
80 //    Ping_measure(3);
81   }
82 }
83
84 //initialize PB4-0
85 //PB4 set as output to send 5us pulse to all four Ping))) sensors at
     same time
86 //PB3-0 set as input to capture input from sensors
87 void Ping_Init(void){
88   /***************** New Style *********************/
89   SYSCTL_RCGCGPIO_R |= SYSCTL_RCGCGPIO_R1;
90   /*************************************************/
91
92   LastStatus = 0;           // (b) initialize status
93
94   GPIO_PORTB_DIR_R &= ~PB3_0;     // (c) make PB3-0 in
95   GPIO_PORTB_AFSEL_R &= ~PB3_0;   //     disable alt funct on PB4-0
96   GPIO_PORTB_DEN_R |= PB3_0;      //     enable digital I/O on PB4-0
97   GPIO_PORTB_PCTL_R &= ~0x000FFFFF; // configure PB4-0 as GPIO
98   GPIO_PORTB_AMSEL_R =~PB3_0;     //     disable analog functionality on
       PB
99   GPIO_PORTB_PDR_R |= PB3_0;      //     enable pull-down on PF4-0
100
101  GPIO_PORTB_IS_R &= ~PB3_0;      // (d) PB3-0 is edge-sensitive
102  GPIO_PORTB_IBE_R |= PB3_0;      //      PB3-0 is both edges
103  GPIO_PORTB_ICR_R =  PB3_0;      // (e) clear flag3-0
104  GPIO_PORTB_IM_R |= PB3_0;       // (f) arm interrupt on PB3-0
105
106  NVIC_PRI0_R = (NVIC_PRI0_R&0xFFFF00FF)|0x00004000; // (g) priority 2
107  NVIC_EN0_R |= NVIC_EN0_INT1;  // (h) enable interrupt 1 in NVIC
108
109  Edge_Valid[0] = Edge_Valid[1] /* = Edge_Valid[2] = Edge_Valid[3] */ =
       0;
110
111  OS_InitSemaphore(&Sema4PingIdle, 1);
112
113  OS_InitSemaphore(&Sema4PingResultAvailable[0], 0);
114  OS_InitSemaphore(&Sema4PingResultAvailable[1], 0);
115 //  OS_InitSemaphore(&Sema4PingResultAvailable[2], 0);
116 //  OS_InitSemaphore(&Sema4PingResultAvailable[3], 0);
```

```
117
118   OS_AddThread(Ping_Thread, 128, 1);
119 }
120
121 unsigned char PingValue(unsigned char *mbox, unsigned char pingNum) {
122   OS_bWait(&Sema4PingResultAvailable[pingNum]);
123
124   *mbox = Distance_Result[pingNum];
125
126   return Sensor_fail[pingNum];
127 }
128
129 //Send pulse to four Ping))) sensors
130 //happens periodically by using timer
131 //foreground thread
132 //Fs: about 40Hz
133 //no input and no output
134
135 // TODO! Decouple PingNum into a parameter
136
137 // Must ensure
138
139 static void Ping_measure(unsigned char number){
140   long sr;
141   unsigned char delay_count;
142   static unsigned char bitmask;
143   unsigned long tin;
144
145   OS_bWait(&Sema4PingIdle);
146
147   PingNum = number & 0x03;
148   bitmask = 1 << PingNum;
149   Edge_Valid[PingNum] = 0;
150
151   // Send pulse
152   GPIO_PORTB_IM_R &= ~bitmask;
153   GPIO_PORTB_DIR_R |= bitmask;
154
155   Sensors |= bitmask;
156   //blind-wait
157   for(delay_count=0; delay_count<100; delay_count++);
158   Sensors &= ~bitmask;
159
160   GPIO_PORTB_DIR_R &= ~bitmask;
161   GPIO_PORTB_IM_R |= bitmask;
162
163   OS_Sleep(TimeGap);
164
165   sr = StartCritical();
166   // Wait for response
167   if(Edge_Valid[PingNum]) {
168     unsigned long d;
169     tin = OS_TimeDifference(Finishtime[PingNum],Starttime[PingNum]);
170     d = ((tin*(3310+6*Temperature+5))/16000000); // cm
171     if(d > 255) d = 255;
```

```
172     Distance_Result[PingNum] = MedianFilter(&filter[PingNum], (unsigned
        char) d);
173     //Distance_Result[PingNum] =  (unsigned char) d;
174     Sensor_fail[PingNum] = 0;
175   } else {
176     Sensor_fail[PingNum] = 1;
177   }
178
179   EndCritical(sr);
180
181   OS_bSignal(&Sema4PingResultAvailable[PingNum]);
182   OS_bSignal(&Sema4PingIdle);
183 }
184
185 //put inside PORTB_handler
186 //input system time, resolution: 12.5ns
187 //no output
188
189 void GPIOPortB_Handler(void){
190   unsigned long CurrStatus = Sensors;
191
192   //check rising edge and record time
193   if(CurrStatus & ~LastStatus) {
194     Starttime[PingNum] = OS_Time();
195   }
196
197   //check falling edge and record time
198   else if(~CurrStatus & LastStatus) {
199     Finishtime[PingNum] = OS_Time();
200
201     Edge_Valid[PingNum] = 1;
202   }
203
204   GPIO_PORTB_ICR_R = PB3_0;
205
206   LastStatus = CurrStatus;
207 }
```

code/ping.c

(c) High-level competition algorithm

Our final competing algorithm employs a PI controller and a finite state machine.

```
1  static long CurrentSpeedR = 0;
2  static long CurrentSpeedL = 0;
3  static long RefSpeedR = 0;
4  static long RefSpeedL = 0;
5
6  static long FrontSideError = 0, LastFrontSideError = 0,
       FrontSideErrorDiff = 0;
7  static long SideError = 0, LastSideError = 0, SideErrorDiff = 0;
8  unsigned char SensorF, SensorFPing, SensorR, SensorL, SensorFR,
       SensorFL;
9
10 typedef enum State_t {GoForward, Stop, SteerRight, SteerLeft,
       GoBackWard, GoStraight} State;
```

```
11
12 void Controller(void) {
13   static int Time = 0, i = 0;
14   static State currentState = GoForward;
15   static int counter = 0;
16   static long error;
17
18   if (Time == 18000) {
19     Motor_Stop();
20     return;
21   } else {
22     Time++;
23   }
24
25   #define Fast_Speed    24000
26   #define Slow_Speed    12000
27   #define Steer_Diff    2000
28   #define Speed_lowbound 5000
29   #define Steering_Forward_P   200
30   #define Steering_Forward_I   10 // Smaller = I term greater
31   #define Sterring_Integral_Capacity 40000
32
33   #define F_Go2Stop_THRS   30
34   #define F_Turn2Go_THRS   F_Go2Stop_THRS+5
35   #define FS_Go2Stop_THRS  8
36   #define FS_Steer2Go_THRS FS_Go2Stop_THRS+5
37
38   #define FRONT            SensorF
39
40   //currentState=TurnLeft;
41   //currentState = GoForward;
42   switch(currentState) {
43     static long error_i = 0;
44     static long diff_error = 0;
45     case GoForward:
46       /*************************************/ Debug_LED(RED);
47
48       RefSpeedL = RefSpeedR = Fast_Speed;
49
50     // + > biasing to right
51     // - < biasing to left
52       error = (SideError + FrontSideError)/2;
53       diff_error = (SideErrorDiff + FrontSideErrorDiff)/2;
54     // By practical observation: the value of error is in the range
    [-255, 255]
55
56       error_i += error;
57       if (error_i > Sterring_Integral_Capacity) error_i =
    Sterring_Integral_Capacity;
58       if (error_i < -Sterring_Integral_Capacity) error_i = -
    Sterring_Integral_Capacity;
59
60       if (error > 0 ) {
61         RefSpeedL -= error * Steering_Forward_P;
62       } else {
63         RefSpeedR += error * Steering_Forward_P;
```

```
64        }
65
66        if (RefSpeedR < Fast_Speed - Speed_lowbound) RefSpeedR =
    Fast_Speed - Speed_lowbound;
67        if (RefSpeedL < Fast_Speed - Speed_lowbound) RefSpeedL =
    Fast_Speed - Speed_lowbound;
68
69        CurrentSpeedR=RefSpeedR;
70        CurrentSpeedL=RefSpeedL;
71
72        //Sterring
73        if (FRONT < F_Go2Stop_THRS || SensorFR < FS_Go2Stop_THRS ||
    SensorFL < FS_Go2Stop_THRS) {
74          currentState = Stop;
75          counter = 0;
76          error_i = 0; break;
77        }
78        break;
79
80      case Stop:
81        /*************************************/ Debug_LED(BLUE);
82
83        // Stoping the wheels
84        RefSpeedR = RefSpeedL = 0;
85        CurrentSpeedL = CurrentSpeedR = 0;
86
87        if (counter  == 50) {
88          if (FRONT < F_Turn2Go_THRS || SensorFR < FS_Go2Stop_THRS ||
    SensorFL < FS_Go2Stop_THRS ) {
89            if (SensorR > SensorL + 5) {
90              currentState=SteerRight;
91            } else if (SensorL > SensorR + 5) {
92              currentState=SteerLeft;
93            } else if (SensorFL < SensorFR) {
94              currentState=SteerRight;
95            } else {
96              currentState=SteerLeft;
97            }
98          } else {
99            currentState = GoForward;
100         }
101         counter = 0;
102       } else {
103         counter ++;
104       }
105       break;
106
107     case SteerRight:
108       /*************************************/ Debug_LED(GREEN);
109       if (counter++ == 100) {
110         counter = 0;
111         currentState = GoBackWard;
112       }
113       RefSpeedR = Fast_Speed/2;
114       RefSpeedL = Fast_Speed;
115
```

```
116       CurrentSpeedR = RefSpeedR;
117       CurrentSpeedL = RefSpeedL;
118
119       // State change
120       if (FRONT > F_Turn2Go_THRS && SensorFR > FS_Steer2Go_THRS &&
    SensorFL > FS_Steer2Go_THRS ){
121         currentState = GoStraight; counter = 0;
122       }
123       break;
124
125     case SteerLeft:
126       /*************************************/ Debug_LED(PURPLE);
127       if (counter++ == 100) {
128         counter = 0;
129         currentState = GoBackWard;
130       }
131       RefSpeedR = Fast_Speed;
132       RefSpeedL = Fast_Speed/2;
133
134       CurrentSpeedR = RefSpeedR;
135       CurrentSpeedL = RefSpeedL;
136
137       // State change
138       if (FRONT > F_Turn2Go_THRS && SensorFR > FS_Steer2Go_THRS &&
    SensorFL > FS_Steer2Go_THRS ) {
139         currentState = GoStraight; counter = 0;
140       }
141       break;
142
143     case GoBackWard:
144       /*************************************/ Debug_LED(VIOLET);
145       if (counter++ == 50) {
146         counter = 0;
147         currentState = GoForward;
148       }
149       RefSpeedR = -Fast_Speed/2;
150       RefSpeedL = -Fast_Speed/2;
151
152       CurrentSpeedR = RefSpeedR;
153       CurrentSpeedL = RefSpeedL;
154       break;
155
156       case GoStraight:
157       /*************************************/ Debug_LED(WHITE);
158       if (counter++ == 60) {
159         counter = 0;
160         currentState = GoForward;
161       }
162       RefSpeedR = Fast_Speed;
163       RefSpeedL = Fast_Speed;
164
165       CurrentSpeedR = RefSpeedR;
166       CurrentSpeedL = RefSpeedL;
167
168       //Sterring
169       if (FRONT < F_Go2Stop_THRS || SensorFR < FS_Go2Stop_THRS ||
```

```
    SensorFL < FS_Go2Stop_THRS) {
170         if (SideError < -5) {
171           currentState = SteerRight;
172         } else if (SideError > 5) {
173           currentState = SteerLeft;
174         } else if (FrontSideError > 0) {
175           currentState = SteerLeft;
176         } else {
177           currentState = SteerRight;
178         }
179         counter = 0;
180         error_i = 0; break;
181       }
182       break;
183
184   }
185
186   Motor_MotionUpdate(CurrentSpeedR, CurrentSpeedL);
187 }
```

code/algorithm.c

(d) Final data flow graph
See Figure-6

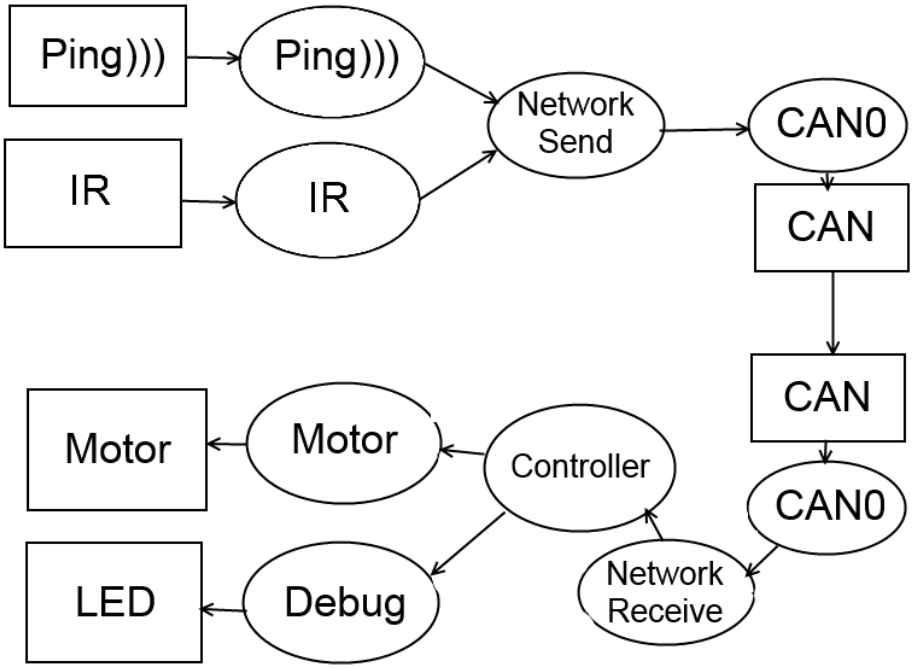

Figure 6: Data flow graph

(d) Final call graph
See Figure-7

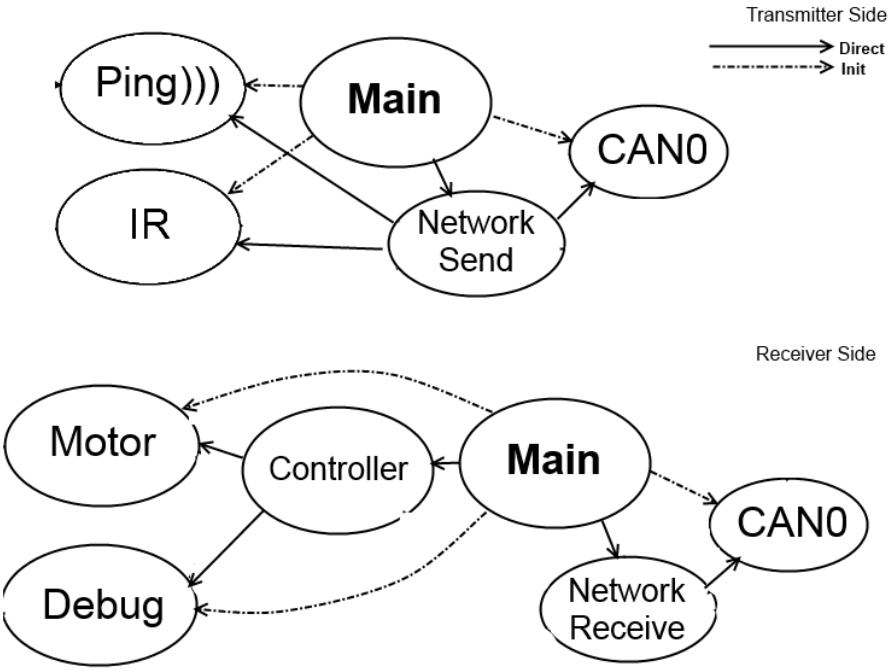

Figure 7: Call graph

Scores		
Run	Attempt	Score
Qualification	1	3
	2	17
	3	8
	4	10
Competition	1	15
	2	15
	3	19

Table 1: Scores

4 Measurement

Our score during the qualification run and final competition is as shown in Table-1.

5 Analysis

(1) What is the effect of time delay in your control system?

The effect of time delay would be that robot reacts to the environment data that belongs to a previous point in time. This could result in robot not turning quickly as soon as it finds a gap, the robot not being able to stabilize its direction among the sides, and the robot hitting the wall before knowing that it has to stop.

(2) What sensors would you need to develop a more effective passing strategy?

IR sensors are more suitable. The most important reason is that IR sensors work well regardless of the angle of the sensor with respective to the object. In this case, since the robot could have any angle to the surrounding walls, IR sensors are the better options compared to ping. Also, IR sensors can provide distance information with lower latency.

(3) If you hit the wall a lot, how could you have changed the design to be more effective? If your robot can travel 3 milestones without hitting a wall, you can skip this question.

Our robot passes 3 milestones without hitting a wall.

(4) Briefly explain how odometry could be used in your robot (we discussed it in class). If you used it, how well did it work and how could it be improved? If you didnt use it, why didnt you use it?

Odometry can be used to estimate the position and orientation of the robot based on its current position. We decided to not use odometry because of two reasons:

- First, odometry requires accurate information of speed and angular velocity. Since we were not using any type of speed sensors, estimating velocity would not result in accurate measurements.
- Second, for the odometry to be effective, an accurate information of the map and environment is required. The prediction of the robot state is not going to be useful if information about the map is not available.

6 Post-Mortem Team Evaluation

Here we evaluate the Strengths and Weaknesses by teammate.

6.1 Chen Cui

Chen	not obvious in this lab; write code slowly
YKH	Knowledge in C and embedded system; unknown
MQ	perfectionist and experience in C; unknown
Siavash	knowledge and experience; unknown
ZY	knowledge in mechanical and hard-working; unknown

6.2 Yen-Kai Huang

Chen	Hardworking and experienced in motor interfacing; None
YKH	Experience in Embedded system and LaTeX; Not very concentrated
MQ	Hardworking and easy to work with; Code style is not professional
Siavash	Excellent embedded programmer; He is often busy with the senior design project
ZY	Very experienced in mechanical engineering and ability of hands-on work ; Lack of embedded experience

6.3 Miao Qi

Chen code is easy readable and in good style, good lab partner; None

YKH His code is stylish and easy-readable, strong leadership; None

MQ easy to communicate; code is not very portable

Siavash Extremely strong coding skill, good organization skill; None

ZY hard working, like to explore deeply; None

6.4 Siavash Zangeneh Kamali

Chen Hardware interface, embedded systems; Not clear and concise in coding style

YKH Precise and clear coding style, good at programming, experienced in embedded systems ; Too much perfectionist

MQ on-time, hard working; Not clear and concise in coding style

Siavash good at programming, experienced in embedded systems ; Doesn't do things until the deadline is written

ZY Mechanical engineering stuff, hard working; Not clear and concise in coding style

6.5 Yan Zhang

Chen 1. expert in the lab 2. working hard ; (It's impossible for me to see you guys' weakness...)

YKH 1. expert in the lab 2. working hard ; (It's impossible for me to see you guys' weakness...)

MQ 1. expert in the lab 2. working hard ; (It's impossible for me to see you guys' weakness...)

Siavash 1. expert in the lab 2. working hard ; (It's impossible for me to see you guys' weakness...)

ZY 1. never give up trying 2. working hard ; 1. little background in EE

Failure and Success in Communication

Thanks to the early set-up with internet tool and code repository, we have had few problems in communication. An added difficulty surfaces when Nick's cell phone broke and so our smartphone app chatroom can no longer be used and we had to resort to using Facebook or face-to-face communication more.