EE445M Lab 6 Report

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

April 8, 2014

1 Objective

The goal of this lab is to prepare for the final Robot race. In this lab we will interface various components, including Ping)))) ultrasonic distance sensor, IR distance sensor, and DC motor. A layered communication system using CAN protocol will transmit the information between two microcontrollers. We will design and implement a software communication protocol to transmit the sensor data and other things.

In this lab we will also form a team of 4 or 5, which requires us also to apply communication skills to function as a team.

2 Hardware Design

Ping sensor is shown at Figure-1

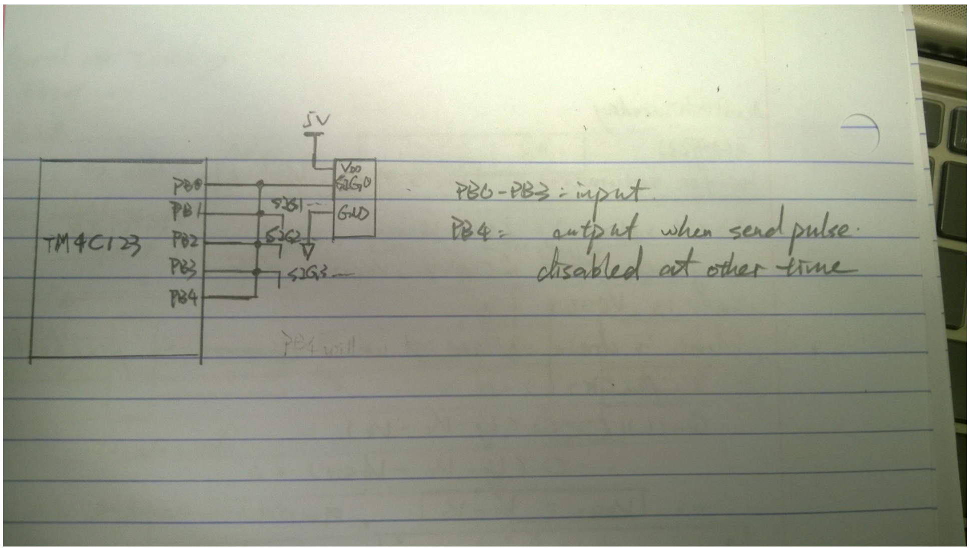

Figure 1

IR sensor is shown at Figure-2
The DC motor circuit is shown at Figure-3

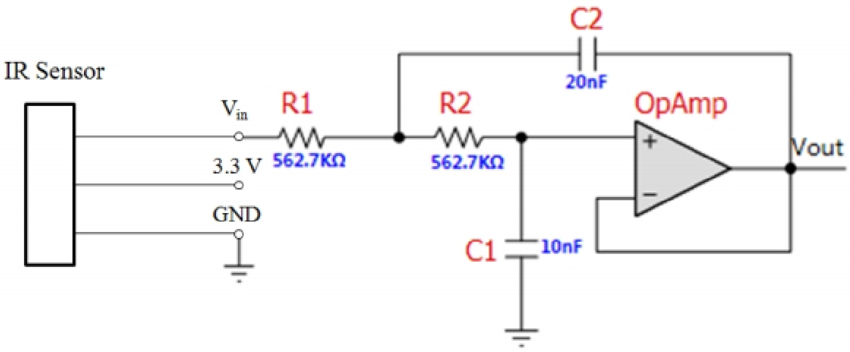

Figure 2

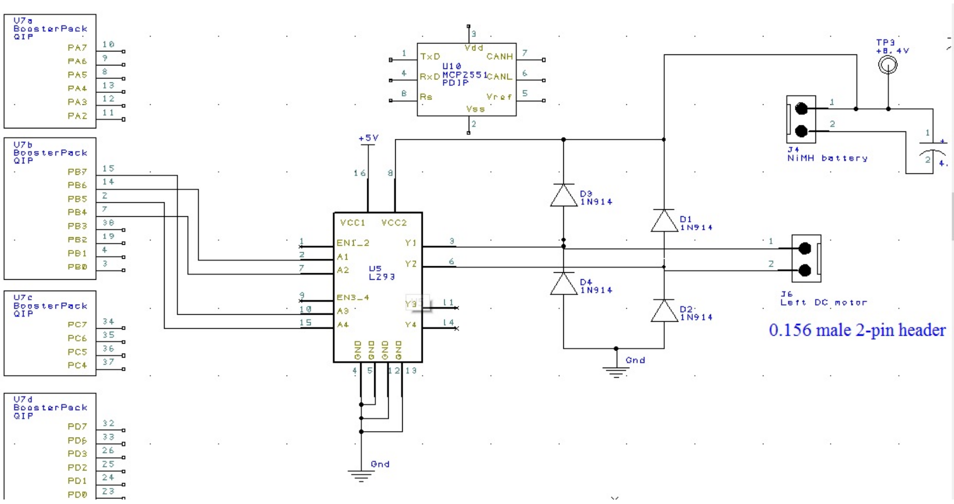

Figure 3

3 Software Design

(a) We made small changes to `CAN0.c`, replacing the spinlock mechanism with actual semaphore.

```
// can0.c
// Runs on LM4F120/TM4C123
// Use CAN0 to communicate on CAN bus PE4 and PE5
//

// Jonathan Valvano
// March 22, 2014

/* This example accompanies the books
   Embedded Systems: Real-Time Operating Systems for ARM Cortex-M
   Microcontrollers, Volume 3,
   ISBN: 978-1466468863, Jonathan Valvano, copyright (c) 2013

   Embedded Systems: Real Time Interfacing to ARM Cortex M
   Microcontrollers, Volume 2
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2013

 Copyright 2014 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/
 */
// MCP2551 Pin1 TXD  ---- CAN0Tx PE5 (8) O TTL CAN module 0 transmit
// MCP2551 Pin2 Vss  ---- ground
// MCP2551 Pin3 VDD  ---- +5V with 0.1uF cap to ground
// MCP2551 Pin4 RXD  ---- CAN0Rx PE4 (8) I TTL CAN module 0 receive
// MCP2551 Pin5 VREF ---- open (it will be 2.5V)
// MCP2551 Pin6 CANL ---- to other CANL on network
// MCP2551 Pin7 CANH ---- to other CANH on network
// MCP2551 Pin8 RS   ---- ground, Slope-Control Input (maximum slew
   rate)
// 120 ohm across CANH, CANL on both ends of network

#include "hw_can.h"
#include "hw_ints.h"
#include "hw_memmap.h"
#include "hw_types.h"
#include "can.h"
#include "debug.h"
#include "interrupt.h"

#include "os.h"
#include "semaphore.h"
```

```c

#include "can0.h"
#include "inc/tm4c123gh6pm.h"


#define NULL 0
// reverse these IDs on the other microcontroller

// Mailbox linkage from background to foreground
PackageID static RCVID;
unsigned char static RCVData[4];
int static MailFlag;

//*************************************************************
//
// The CAN controller interrupt handler.
//
//*************************************************************
void CAN0_Handler(void){ unsigned char data[4];
  unsigned long ulIntStatus, ulIDStatus;
  int i;
  tCANMsgObject xTempMsgObject;
  xTempMsgObject.pucMsgData = data;
  ulIntStatus = CANIntStatus(CAN0_BASE, CAN_INT_STS_CAUSE); // cause?
  if(ulIntStatus & CAN_INT_INTID_STATUS){  // receive?
    ulIDStatus = CANStatusGet(CAN0_BASE, CAN_STS_NEWDAT);
    for(i = 0; i < 32; i++){    //test every bit of the mask
      if( (0x1 << i) & ulIDStatus){  // if active, get data
        CANMessageGet(CAN0_BASE, (i+1), &xTempMsgObject, true);
        //if(xTempMsgObject.ulMsgID == RCV_ID){
        RCVID = (PackageID) xTempMsgObject.ulMsgID;
        RCVData[0] = data[0];
        RCVData[1] = data[1];
        RCVData[2] = data[2];
        RCVData[3] = data[3];
        //MailFlag = true;   // new mail
        OS_bSignal(&Sema4CAN);
        //}
      }
    }
  }
  CANIntClear(CAN0_BASE, ulIntStatus);  // acknowledge
}

//Set up a message object.  Can be a TX object or an RX object.
void static CAN0_Setup_Message_Object( unsigned long MessageID, \
                                unsigned long MessageFlags, \
                                unsigned long MessageLength, \
                                unsigned char * MessageData, \
                                unsigned long ObjectID, \
                                tMsgObjType eMsgType){
  tCANMsgObject xTempObject;
  xTempObject.ulMsgID = MessageID;          // 11 or 29 bit ID
  xTempObject.ulMsgLen = MessageLength;
  xTempObject.pucMsgData = MessageData;
  xTempObject.ulFlags = MessageFlags;
```

```
  CANMessageSet(CAN0_BASE, ObjectID, &xTempObject, eMsgType);
}
// Initialize CAN port
void CAN0_Open(void){unsigned long volatile delay;

  MailFlag = false;
  OS_InitSemaphore(&Sema4CAN, 0);

  SYSCTL_RCGCCAN_R |= 0x00000001;  // CAN0 enable bit 0
  SYSCTL_RCGCGPIO_R |= 0x00000010;  // RCGC2 portE bit 4
  for(delay=0; delay<100; delay++){};
  GPIO_PORTE_AFSEL_R |= 0x30; //PORTE AFSEL bits 5,4
// PORTE PCTL 88 into fields for pins 5,4
  GPIO_PORTE_PCTL_R = (GPIO_PORTE_PCTL_R&0xFF00FFFF)|0x00880000;
  GPIO_PORTE_DEN_R |= 0x30;
  GPIO_PORTE_DIR_R |= 0x20;

  CANInit(CAN0_BASE);
  CANBitRateSet(CAN0_BASE, 80000000, CAN_BITRATE);
  CANEnable(CAN0_BASE);
// make sure to enable STATUS interrupts
  CANIntEnable(CAN0_BASE, CAN_INT_MASTER | CAN_INT_ERROR | CAN_INT_STATUS);
// Set up filter to receive these IDs
// in this case there is just one type, but you could accept multiple ID types
  //CAN0_Setup_Message_Object(RCV_ID, MSG_OBJ_RX_INT_ENABLE, 4, NULL, RCV_ID, MSG_OBJ_TYPE_RX);
  CAN0_Setup_Message_Object((unsigned long) (IRSensor0), MSG_OBJ_RX_INT_ENABLE, 4, NULL, (unsigned long) (IRSensor0), MSG_OBJ_TYPE_RX);
  CAN0_Setup_Message_Object((unsigned long) (UltraSonic), MSG_OBJ_RX_INT_ENABLE, 4, NULL, (unsigned long) (UltraSonic), MSG_OBJ_TYPE_RX);
  NVIC_EN1_R = (1 << (INT_CAN0 - 48)); //IntEnable(INT_CAN0);
  return;
}

// send 4 bytes of data to other microcontroller
void CAN0_SendData(PackageID sendID, unsigned char data[4]){
// in this case there is just one type, but you could accept multiple ID types
  CAN0_Setup_Message_Object((unsigned long) sendID, NULL, 4, data, (unsigned long) sendID, MSG_OBJ_TYPE_TX);
}

// Returns true if receive data is available
//         false if no receive data ready
int CAN0_CheckMail(void){
  return MailFlag;
}
// if receive data is ready, gets the data and returns true
// if no receive data is ready, returns false
/*****Not implemented
int CAN0_GetMailNonBlock(unsigned char data[4]){
  if(MailFlag){
```

```
    data[0] = RCVData[0];
    data[1] = RCVData[1];
    data[2] = RCVData[2];
    data[3] = RCVData[3];
    MailFlag = false;
    return true;
  }
  return false;
}
*/

// if receive data is ready, gets the data
// if no receive data is ready, it waits until it is ready
void CAN0_GetMail(PackageID *receiveID, unsigned char data[4]){
  OS_bWait(&Sema4CAN);
  *receiveID = RCVID;
  data[0] = RCVData[0];
  data[1] = RCVData[1];
  data[2] = RCVData[2];
  data[3] = RCVData[3];
}
```

code/can0.c

We also modified the FIFO macro to internally include semaphores.

```
// FIFO.h
// Runs on any LM3Sxxx
// Provide functions that initialize a FIFO, put data in, get data out,
// and return the current size.  The file includes a transmit FIFO
// using index implementation and a receive FIFO using pointer
// implementation.  Other index or pointer implementation FIFOs can be
// created using the macros supplied at the end of the file.
// Daniel Valvano
// June 16, 2011

// April 2, 2014
// Modified
// - Added semaphore
// Nick Huang

/* This example accompanies the book
   "Embedded Systems: Real Time Interfacing to the Arm Cortex M3",
   ISBN: 978-1463590154, Jonathan Valvano, copyright (c) 2011
      Programs 3.7, 3.8., 3.9 and 3.10 in Section 3.7

 Copyright 2011 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/
 */

// macro to create an index FIFO
#define AddIndexSema4Fifo(NAME,SIZE,TYPE,SUCCESS,FAIL) \
Sema4Type Sema4 ## NAME;                             \
unsigned long volatile NAME ## PutI;     \
unsigned long volatile NAME ## GetI;     \
TYPE static NAME ## Fifo [SIZE];         \
void NAME ## Fifo_Init(void){ long sr;   \
  sr = StartCritical();                  \
  NAME ## PutI = NAME ## GetI = 0;       \
  EndCritical(sr);                       \
  OS_InitSemaphore(&Sema4 ## NAME, 0);   \
}                                        \
int NAME ## Fifo_Put (TYPE data){        \
  if(( NAME ## PutI - NAME ## GetI ) & ~(SIZE-1)){  \
    return(FAIL);       \
  }                     \
  NAME ## Fifo[ NAME ## PutI &(SIZE-1)] = data; \
  NAME ## PutI ## ++;   \
  OS_Signal(&Sema4 ## NAME); \
  return(SUCCESS);      \
}                       \
int NAME ## Fifo_Get (TYPE *datapt){  \
  OS_Wait(&Sema4 ## NAME); \
  if( NAME ## PutI == NAME ## GetI ){ \
    return(FAIL);       \
  }                     \
  *datapt = NAME ## Fifo[ NAME ## GetI &(SIZE-1)];  \
  NAME ## GetI ## ++;   \
  return(SUCCESS);      \
}                       \
unsigned short NAME ## Fifo_Size (void){  \
 return ((unsigned short)( NAME ## PutI - NAME ## GetI ));  \
}

#define AddCallBackFunction(NAME) \
long NAME ## DataLost;            \
void NAME ## CallBack(unsigned short ADCvalue) { \
  if (!NAME ## Fifo_Put(ADCvalue)){            \
    NAME ## DataLost ++;                            \
  }                                            \
}
```

code/FIFO_sema4.h

(b) IR sensor code is basically the same as Lab. 4, except that I decoupled the `Filter()` functions from the filter buffer. The call-back functions in ADC routine is now responsible for acquiring and storing the data.

```
// IR_sensor.h

#ifndef __IR_SENSOR_H__
#define __IR_SENSOR_H__
```

```c

#define LAB_DEMO 6

void IR_Init(void);
void IR_getValues (unsigned short *buffer);

#endif // __IR_SENSOR_H__
```

code/ir_sensor.h

```c
// IR_sensor.c

#include "ir_sensor.h"

#include "OS.h"
#include "FIFO_sema4.h"
#include "adc.h"

#define SAMPLING_RATE 2000

/********************** Data Structure ******************/
#define FIFOSIZE   64         // size of the FIFOs (must be power of 2)
#define FIFOSUCCESS 1         // return value on success
#define FIFOFAIL    0         // return value on failure
                              // create index implementation FIFO (see
    FIFO.h)
  long StartCritical(void);    // previous I bit, disable interrupts
  void EndCritical(long sr);    // restore I bit to previous value

AddIndexSema4Fifo(IR1, FIFOSIZE, unsigned short, FIFOSUCCESS, FIFOFAIL)
AddCallBackFunction(IR1)

#if LAB_DEMO == 7
AddIndexSema4Fifo(IR2, FIFOSIZE, unsigned short, FIFOSUCCESS, FIFOFAIL)
AddCallBackFunction(IR2)
AddIndexSema4Fifo(IR3, FIFOSIZE, unsigned short, FIFOSUCCESS, FIFOFAIL)
AddCallBackFunction(IR3)
AddIndexSema4Fifo(IR4, FIFOSIZE, unsigned short, FIFOSUCCESS, FIFOFAIL)
AddCallBackFunction(IR4)
#endif

/*********************** Filter ***************************/
#define FILTER_LENGTH 51
const long ScaleFactor = 16384;
const long H[51]={-11,10,9,-5,1,0,-19,6,48,-12,-92,
     17,155,-20,-243,22,370,-24,-559,24,881,-24,-1584,24,4932,
     8578,4932,24,-1584,-24,881,24,-559,-24,370,22,-243,-20,155,
     17,-92,-12,48,6,-19,0,1,-5,9,10,-11};

typedef struct {
  // this MACQ needs twice the size of FILTER_LENGTH
  long x[2*FILTER_LENGTH];
  unsigned char index;
} FilterType;

static unsigned short IRsensor1;
```

```c

static FilterType filter1 = {{0},FILTER_LENGTH-1};

// Filter
// Digital FIR filter, assuming fs=1 Hz
// Coefficients generated with FIRdesign64.xls
// y[i]= (h[0]*x[i]+h[1]*x[i-1]+   +h[63]*x[i-63])/256;
static unsigned short Filter(FilterType *f, unsigned short data) {
  long y = 0;
  unsigned char i;

  if(++f->index == 2*FILTER_LENGTH) f->index = FILTER_LENGTH;
  f->x[f->index] = f->x[f->index-FILTER_LENGTH] = data;

  // Assuming there is no overflow
  for(i = 0; i < FILTER_LENGTH; ++i){
    y += H[i]*f->x[f->index-i];
  }
  y /= ScaleFactor;

  return y;
}

/****************************************************/

// Consumer
// Foreground thread that takes in data from FIFO, apply filter, and
   record data
// If trigger Capture is set, it will perform a 64-point FFT on the
   recorded data
// and store result in fft_output[]
// Block when the FIFO is empty
#define NOW_USING   1
#define STOP_USING  0
#define NOT_USING  -1
char Filter_Use = NOT_USING;

static void Consumer(void) {
  ADC_Collect(0, SAMPLING_RATE, IR1CallBack, 64);

  while (1) {
    // Get data, will block if FIFO is empty
    unsigned short data;
    IR1Fifo_Get(&data);

    // Choosing whether to apply the filter
    IRsensor1 = Filter(&filter1, data);
  }
}

void IR_Init(void) {
  OS_InitSemaphore(&Sema4DataAvailable, 0);

  IR1Fifo_Init();

  OS_AddThread(&Consumer, 256, 3);
```

```
}

void IR_getValues (unsigned short *buffer) {
  buffer[0] = IRsensor1;
  #if LAB_DEMO == 7
  buffer[1] = IRsensor2;
  buffer[2] = IRsensor3;
  buffer[3] = IRsensor4;
  #endif
}
```

code/ir_sensor.c

For the Ping))) interfacing, we used one pin to output the 5 μs pulse, and one pin for each sensor set to trigger an interrupt at both edges. The handler will calculate the difference of time between the two edges and thus determine the distance measurement.

```
// Ping.h
// Runs on LM4C123
// Initialize Ping interface, then generate 5us pulse about 10 times
    per second
// capture input pulse and record pulse width
// Miao Qi
// October 27, 2012

//initialize PB4-0
//PB4 set as output to send 5us pulse to all four Ping))) sensors at
    same time
//PB3-0 set as input to capture input from sensors
void Ping_Init(void);

void Ping_getData(unsigned long * data) ;
```

code/ping.h

```
// Ping.c
// Runs on LM4C123
// Initialize Ping interface, then generate 5us pulse about 10 times
    per second
// capture input pulse and record pulse width
// Miao Qi
// October 27, 2012

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

#define PB4            (*((volatile unsigned long *)0x40005040))
#define PB3_0            (*((volatile unsigned long *)0x4000503C))
#define Temperature        20
#define NVIC_EN0_INT1      2

#define TIME_1MS 80000

unsigned long Ping_Lasttime[4];
unsigned long Ping_Finishtime[4];
unsigned char Ping_Update;
unsigned long Ping_Distance_Result[4];
```

```c
unsigned long Ping_Distance_Filter[4][4];
//unsigned long Ping_Distance_cal[10];
unsigned long Ping_Index[4];
unsigned long Ping_laststatus;

void Ping_pulse(void);

//initialize PB4-0
//PB4 set as output to send 5us pulse to all four Ping))) sensors at same time
//PB3-0 set as input to capture input from sensors
void Ping_Init(void){
                                // (a) activate clock for port F
  SYSCTL_RCGC2_R |= SYSCTL_RCGC2_GPIOB;
  Ping_laststatus = 0;             // (b) initialize status
  GPIO_PORTB_DIR_R &= ~0x0F;    // (c) make PB3-0 in
  GPIO_PORTB_DIR_R |=  0x10;    // (c) make PB4 out
  GPIO_PORTB_AFSEL_R &= ~0x1F;  //     disable alt funct on PB4-0
  GPIO_PORTB_DEN_R |= 0x1F;     //     enable digital I/O on PB4-0
  GPIO_PORTB_PCTL_R &= ~0x000FFFFF; // configure PB4-0 as GPIO
  GPIO_PORTB_AMSEL_R = 0;       //     disable analog functionality on PB
  GPIO_PORTB_PDR_R |= 0x1F;     //     enable pull-down on PF4-0
  GPIO_PORTB_IS_R &= ~0x0F;     // (d) PB3-0 is edge-sensitive
  GPIO_PORTB_IBE_R |= 0x0F;    //      PB3-0 is both edges
  GPIO_PORTB_ICR_R =  0x0F;      // (e) clear flag3-0
  GPIO_PORTB_IM_R |= 0x0F;      // (f) arm interrupt on PB3-0
  NVIC_PRI0_R = (NVIC_PRI0_R&0xFFFF00FF)|0x00004000; // (g) priority 2
  NVIC_EN0_R |= NVIC_EN0_INT1;  // (h) enable interrupt 1 in NVIC
  OS_AddPeriodicThread(&Ping_pulse, 100*TIME_1MS, 3);
}

extern unsigned char SendPulse;
extern unsigned long PulseCount;
//Send pulse to four Ping))) sensors
//happens periodically by using timer
//foreground thread
//Fs: about 10Hz
//no input and no output

void Ping_pulse(void){
unsigned char delay_count;
  GPIO_PORTB_DEN_R |= 0x10;
  GPIO_PORTB_DEN_R &= ~0x0F;
  PB4 = 0x10;
  //blind-wait
  for(delay_count=0; delay_count<60; ){delay_count++;}
  PB4 = 0x00;
  GPIO_PORTB_DEN_R &= ~0x10;
  GPIO_PORTB_DEN_R |=  0x0F;
}

unsigned long median(unsigned long *data_record){
unsigned long buffer[4];
//compare the oldest two data
if((*data_record)<*(data_record+1))
```

```
  {buffer[0]=*data_record; buffer[1]=*(data_record+1);}
else
  {buffer[1]=*data_record; buffer[0]=*(data_record+1);}
//compare the third data
if(buffer[0]<*(data_record+2)){
  if(buffer[1]<*(data_record+2)){buffer[2]=*(data_record+2);}
  else{buffer[2]=buffer[1]; buffer[1]=*(data_record+2);}
  }
else{buffer[2]=buffer[1]; buffer[1]=buffer[0];buffer[0]=*(data_record
    +2);}
//compare the forth data
//ingore the forth data when it is the laragest
if(buffer[2]>*(data_record+3)){
    //ingore the forth data when it is the smallest
    if(buffer[0]>*(data_record+3)){buffer[2]=buffer[1];buffer[1]=buffer
    [0];}}
    else{buffer[2]=*(data_record+3);}
return (buffer[1]+buffer[2])>>1;
}



//d=c* tIN/2
//d = c * tIN * 12.5ns /2 * (um/us)
//d = c * tIN * (1us/40) /(2*2) * (um/us)
//d = c * tIN / (40*2*2) * um
//ignore underflow
//+0.5: round
//return distance = ((tin/40)*(331+0.6*Temperature+0.5))/4;
//compute and update distance array for four sensors
//called when PORTB3-0 capture a value change
//output resolution um
void Distance(void){
unsigned char bits_I = 0;
unsigned long tin;
for (bits_I=0; bits_I<4;bits_I++)
  if(Ping_Update&(1<<bits_I)) {
    tin = OS_TimeDifference(Ping_Finishtime[bits_I],Ping_Lasttime[
    bits_I]);
    tin = ((tin/40)*(331+0.6*Temperature+0.5))/4;
    Ping_Distance_Filter[bits_I][Ping_Index[bits_I]&0x3] = tin;
    Ping_Index[bits_I]++;
    Ping_Distance_Result[bits_I] = median(&Ping_Distance_Filter[bits_I
    ][0]);
  //  Ping_Distance_Result[bits_I] = tin//80000;
    Ping_Update &= ~(1<<bits_I);
  }
}


//put inside PORTB_handler
//input system time, resolution: 12.5ns
//no output
void GPIOPortB_Handler(void){
//void Ping_measure(void){
  unsigned char bits_I = 0;
```

```
  unsigned long Ping_status;
  Ping_status = PB3_0;
  //check rising edge and record time
  for (bits_I=0; bits_I<4;bits_I++) {
    Ping_Lasttime[bits_I] = ((Ping_status&(1<<bits_I)) && !(
   Ping_laststatus&(1<<bits_I)))? OS_Time():Ping_Lasttime[bits_I];
    GPIO_PORTB_ICR_R = 1<<bits_I;
  }
  //check falling edge and compute distance
  for (bits_I=0; bits_I<4;bits_I++) {
    Ping_Finishtime[bits_I] = (!(Ping_status&(1<<bits_I)) && (
   Ping_laststatus&(1<<bits_I)))? OS_Time():Ping_Finishtime[bits_I];
    GPIO_PORTB_ICR_R = 1<<bits_I;
    Ping_Update |= 1<<bits_I;
  }
  Ping_laststatus = Ping_status;
}

void Ping_getData(unsigned long * data) {
  int i;
  Distance();
  for (i=0;i<4;i++) {
    data[i] = Ping_Distance_Result[i];
  }
}
```

code/ping.c

(c) The code that sets up the distributed data acquisition system comes in two sides. On the transmitter side, the `main` initializes the sensors and network, and sends data periodically at about 10 Hz.

```
void NetworkSend(void) {
  unsigned short IRvalues[4];
  unsigned long sonarValues[4];
  unsigned char CanData[4];

  IR_getValues(IRvalues);
  ((unsigned short*)CanData)[0] = IRvalues[0];
  CAN0_SendData(IRSensor0, CanData);

  Ping_getData (sonarValues);
  ((unsigned long*)CanData)[0] = sonarValues[0];
  CAN0_SendData(UltraSonic, CanData);
}

int main(void) {
  PLL_Init();
  OS_Init();

  // Initialize sensors
  IR_Init();
  Ping_Init();

  // Initialize network
  CAN0_Open();

```

```c
  NumCreated += OS_AddPeriodicThread(&NetworkSend, 100*TIME_1MS, 3);

  OS_Launch(TIMESLICE);
}
```

code/main_TX.c

On the receiver side, the `main` initializes the display and network, then add a thread that waits for the packet from the network. Once it receives a packet, it checks the packet ID to determine the type of data and then publish it to the display.

```c
void NetworkReceive(void) {
  PackageID receiveID;
  unsigned char canData[4];

  // Initialize network
  CAN0_Open();

  while (1) {
    CAN0_GetMail(&receiveID, canData);
    switch(receiveID) {
      case IRSensor0:
        ST7735_Message(0,0,"IR0: ", ((unsigned short *)canData)[0]);
        dataReceived++;
      break;
      case UltraSonic:
        ST7735_Message(0,1,"ULS0: ", ((unsigned long *)canData)[0]);
      break;
      default:
      break;
    }
  }
}

int main(void) {
  PLL_Init();
  OS_Init();

  // Initialize Display
  ST7735_InitR(INITR_REDTAB);
  ST7735_SetRotation(1);
  ST7735_FillScreen(0);

  NumCreated += OS_AddThread(&NetworkReceive,128,1);

  OS_Launch(TIMESLICE);
}
```

code/main_RX.c

This is a very simplistic example, however, it contains the full range of function for a distributed Data Acquisition System.

4 Measurement

(a) Ping))) Calibration Ping measurements in Table-1

Ping measurements										
Truth dT(cm)	measured dM(cm)								standard deviation	span
10	10.1538	10.2742	10.129	10.2398	9.9102	10.1231	10.2314	9.9324	0.136715993	0.364
20	20.7653	20.9498	20.4286	20.5348	20.8695	21.0135	20.8493	20.9756	0.212323325	0.5849
30	31.4354	31.2317	31.2342	31.5463	31.8467	31.6756	31.3765	31.0475	0.260240679	0.7992
50	52.7532	52.8654	52.2543	52.7397	52.9485	53.0123	52.9475	53.2397	0.286319955	0.9854
80	83.1323	82.1398	83.2538	84.0132	83.8764	83.1233	82.4956	83.2835	0.626714781	1.8734

Table 1: Ping measurements

IR sensor measurements						
Truth dT(c0)	ADC average	ADC Std. Dev.	ADC span	Voltage average(V)	Voltage Std. Dev.(V)	Voltage span (V)
10	2822	5	19	2.067	0.004	0.014
15	1979	4	15	1.450	0.003	0.011
20	1570	5	19	1.150	0.003	0.014
25	1282	6	25	0.939	0.005	0.018
30	1114	5	27	0.816	0.004	0.020
35	941	2	8	0.689	0.001	0.006
40	848	7	23	0.621	0.005	0.017
45	748	6	24	0.548	0.004	0.018
50	675	5	18	0.494	0.003	0.013

Table 2: IR measurements

(b) IR sensor Calibration IR sensor measurements in Table-2

(c) IR sensor noise spectrum We held a piece of paper constant from the IR sensor and recorded this noise spectrum. Because the measured object is nearly static, everything except the DC component should be considered noise. The noise therefore includes a noise floor that spans all frequencies. The major source of the noise is the thermal noise that includes all frequency and with amplitude of $\frac{1}{f}$. The other source of noise is electrostatics property of the sensor itself, which shows a visible small squarewave in the time domain. This squarewave translates to a noise of all frequency in the frequency domain.

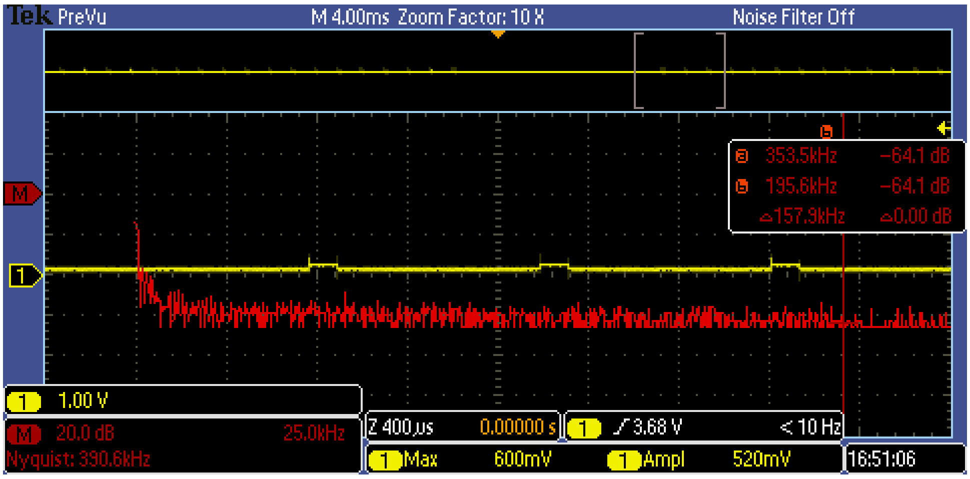

Figure 4: Frequency spectrum of noise

(d) Motor DC Motor measurements in Table-3

(e) CAN scope picture Below is a CAN packet measured at the CANH bus (yellow) and the microcontroller output (blue). It can be observed that the CANH signal is a inverted version of the microcontroller output, however the microcontroller output is changing between 0 and $3.3\,V$, and the CANH signal is changing between about 2.5 and $3.8\,V$. Also, the length of a typical packet is $161.7\,\mu s$

(f) CAN Network bandwidth We set the CAN network bit rate at $1Mbit$ and decrease the sampling period of IR sensor. The result is shown in Table-4.

Therefore we deduct the maximum sampling rate the system can handle is $\frac{1}{5\,ms} \times 4$ bytes = 800 bytes/sec. Considering the packet length measured in Figure-5, CAN network itself should not be the limiting fac-

DC Motor measurements		
Condition	Voltage across motor	Current
no-load	4.41v	32mA
with rubber wheel	4.42v	34mA
wheel & on carpet	4.42v	100mA

Table 3: DC motor measurements

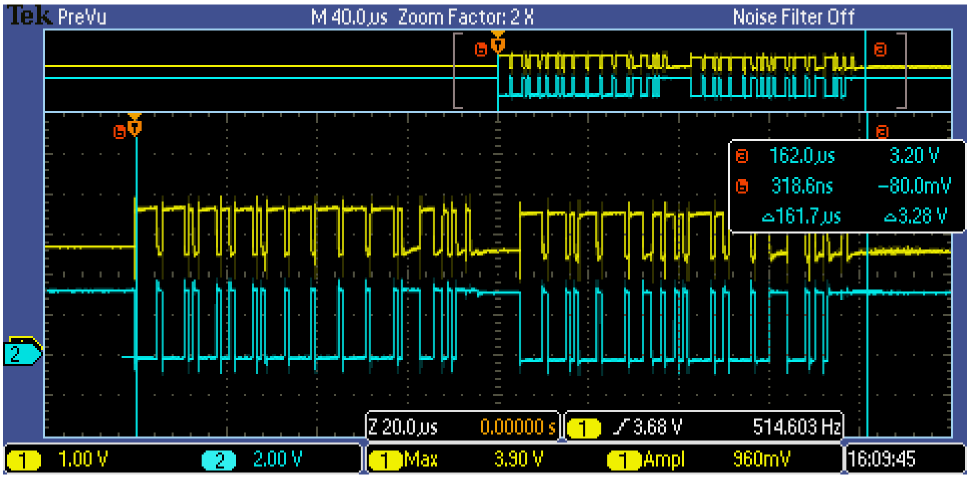

Figure 5: Scope trace of a typical CAN packet

CAN Bandwidth	
Sampling Rate	Data Lost
100 ms	No
10 ms	No
5 ms	No
4 ms	Yes
3 ms	Yes
2 ms	Yes
1 ms	Yes

Table 4: Bandwith measurement

tor. The limiting factor is probably the execution speed of the data acquisition thread on the receiver side, which needs to run a 51-point filter code for each IR sensor data.

5 Analysis

(1) What is one advantage of the Ping))) sensor over the GP2Y0A21YK sensor?

Ping))) outputs a PWM signal which duty cycles changes linearly with respect to the distance, so it's easier to convert the raw data to distance.

(2) What is one advantage of the GP2Y0A21YK sensor over the Ping))) sensor

GP2Y0A21YK measures the distance by an analog voltage. It is easier to convert voltage to digital data using ADC compared to measure time in Ping.

(3) Describe the noise of the GP2Y0A21YK when measured with a spectrum analyzer.

The noise of the sensor is periodic square waves. An square wave contains all different frequencies, with Maximum amplitude at the frequency of the square wave. You can see that the spectrum detects many different frequencies.

(4) Why did you choose the digital filters for your sensors?

What is the time constant for this filter? I.e., if there is a step change in input, how long until your output changes to at least 1/e of the final value?

Since the transition bandwidth of the analog filter is too large for a 2-pole low-pass filter, therefore we use the digital filter to filter out the noise.

An analog filter with higher poles is more expensive.

Since we're using a 51 point FIR filter, and $1/e = 0.36$, we need to sample about 17 points to get enough data to represent 0.36 of the input.

Therefore, time constant $= \frac{17}{f_c} = 8.5\, ms$

(5) Present an alternative design for your H-bridge and describe how your H-bridge is better or worse?

Alternate design is shown as Figure-6

Using our H-bridge is a trade-off. It is better in the simplicity of circuit design so to minimize risk of damaging circuit. Also it saves space. The alternate H-bridge is more sophisticated. This design gets a faster response time of the signal. However it's easier to cause problem and needs more components.

(6) Give the single-most important factor in determining the maximum bandwidth on this distributed system. Give the second-most important factor. Justify your answers.

Since the system bandwidth was much less than the most possible theoretical bandwidth according to bit rate, the most important factor is the amount of time needed to prepare a package to be sent in the CAN driver, and the amount of time to receive and interpret an incoming package.

The second-most important factor is the bit-rate that determines the amount of time it takes for a single package to be transmitted over the bus.

Alternative H-bridge circuit at Figure-6

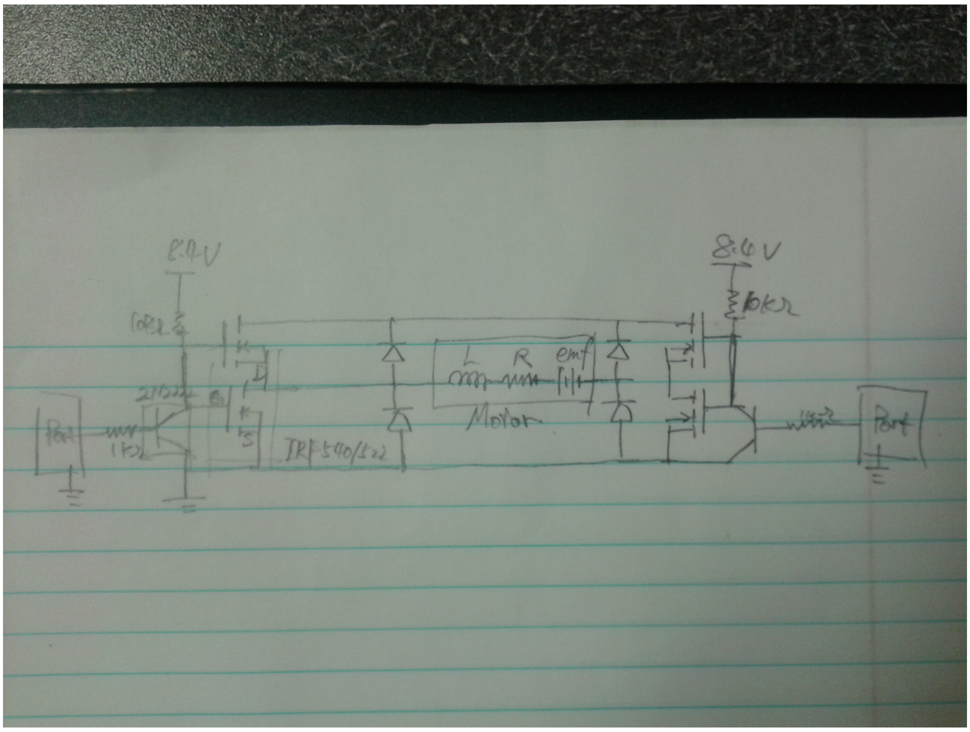

Figure 6

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

6.3 Miao Qi

6.4 Siavash Zangeneh Kamali

6.5 Yan Zhang