ME 586: Refrigeration System

Sesha N. Charla

December 8, 2022

Abstract

A controller for the refrigeration system is designed to maintain the temperature of the fluid at the output of the valve. The temperature drop accross the valve is modelled as the function of valve opening (input-output model) using the step-response. Similarly, the change in temperature accross the valve with compressor speed change is modelled as well. One of the goals of the controller is to reject the disturbances in the temperature drop due to compressor speed. The performance of the designed controller is demostrated.

1 System Setup and Static Calibration

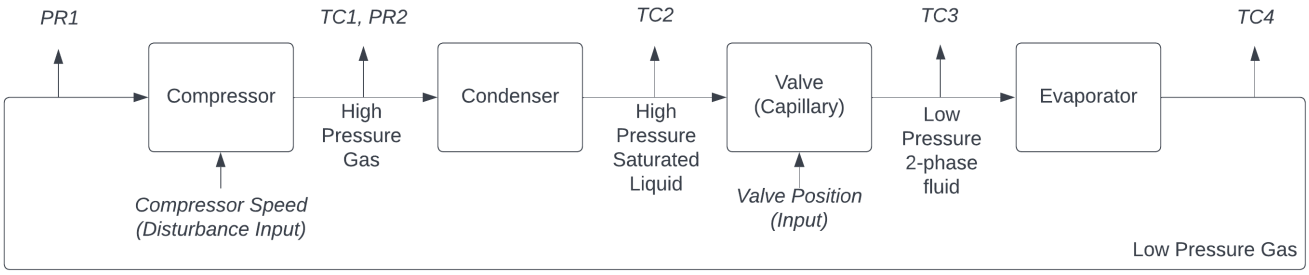


Figure 1: Refrigeration sytem with inputs and outputs of each subsystem

The above figure delineats the interconnections and the sensor locations between the four subsystems of the refrigerator, compressor (disturbance source), condensor, valve (actuator) and evaporator (load). The purpose of valve control is assumed to be maintain the temperature of the fluid entering the evaporator. The justification for such a goal is that this ensures that there is a temperature difference between the load (ambient temperature in present case) and the fluid passing though the evaporator to allow 'efficient' heat transfer.

Thus, the goal of the valve-controller (control objective) is identified as to maintain the temperature TC3 in the presence of disturbance input in the form of change in compressor speed that directly effects TC2 and the temperature drop accross the valve. The valve controls the temperature drop accross it, i.e., TC3-TC2. This is used as the output for the system. The input is the percentage opening of the valve. The disturbance effect of the compressor speed is assumed to be additive to the output (a full dyanmic model of the refrigerator could provide a justification or invalidate this assumption which is out of scope of the present work).

1.1 Static Calibration of Temperature and Pressure Sensors

Compressor Speed: 100 Hz

Valve Volt	Valve %	TC1 Volt	TC1 deg	TC2 Volt	TC2 deg	TC3 Volt	TC3 deg	TC4 Volt	TC4 deg	PR1 Volt	PR1 PSI	PR1 Volt	PR1 PSI
1.224	35	1.484	47.3	1.395	29.3	1.1898	-12	1.281	6.5	0.692	10.5	1.05	70
1.631	43	1.495	49.5	1.396	29.6	1.201	-9.7	1.278	7	1.05	10.8	1.08	104
2.04	51	1.5	51.3	1.396	29.4	1.234	-3.2	1.281	7.8	1.31	10.8	1.08	133
2.454	60	1.51	52.5	1.391	28.6	1.254	1	1.295	9.2	1.554	10.7	1.07	155
2.861	68	1.51	53.2	1.38	27.9	1.27	5	1.3	11	1.74	10.5	1.05	174
3.27	77	1.51	54.3	1.38	27.1	1.29	8.4	1.31	13.1	1.89	10.3	1.03	189

Figure 2: Static Calibration Data

DAC Counts	Voltage	ADC -4 (TC2)	ADC-3 (TC3)
2300	-0.109	2208	2210
2400	0.346	2415	2413
2500	0.804	2621	2620
2600	1.259	2828	2825
2700	1.717	3032	3030
2800	2.172	3240	3237
2900	2.627	3444	3440
3000	3.086	3651	3647
3100	3.542	3853	3851
3200	3.997	4064	4058

Figure 3: DAC Calibration Data

From the calibration experiments, we have the following relavent static calibration equations for input-output data processing:

1. DAC output voltage to DAC voltage:

$$counts = volts \times 219.1978 + 2323.9$$

2. Valve's percentage opening to the voltage:

3. Voltage to the temperature:

$$temperature = \frac{(volts - 1.25)}{0.005}$$

4. ADC-4 (TC2) counts to voltage:

$$volts = 0.0022 \times counts - 5.0047$$

5. ADC-4 (TC3) counts to voltage:

$$volts = 0.0022 \times counts - 5.0179;$$

2 Prelimnary System Identification

Input and disturbance step resoponses are used to identify a first order models (assumption) for input-ouput and disturbance-output systems respectively.

2.1 Valve position step response (Input)

For valve position step, the valve opening percentage is changed from 50% to 30%. The change in valve position is plotted with the change in θ.

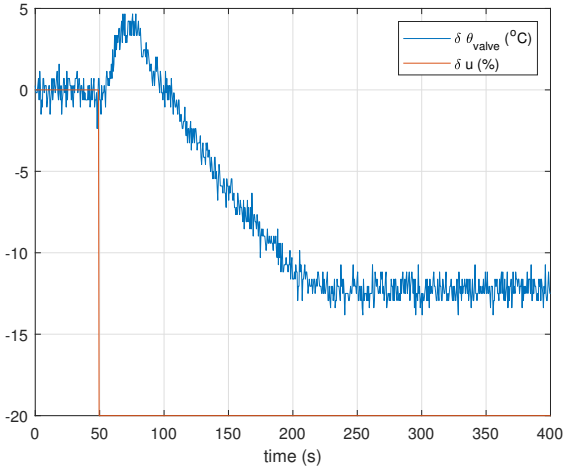


Figure 4: Input Step Response

From the step-resoponse, a second order model with a non-minimum phase (RHP) zero is assumed to be the model structure for the input-output model to account for the undershoot.

The following transfer function model is identified using MATLAB's system identification toolbox.

$$G(s) = \frac{-0.007687s + 8.618 \times 10^{-05}}{s^2 + 0.01878s + 0.0001422}$$

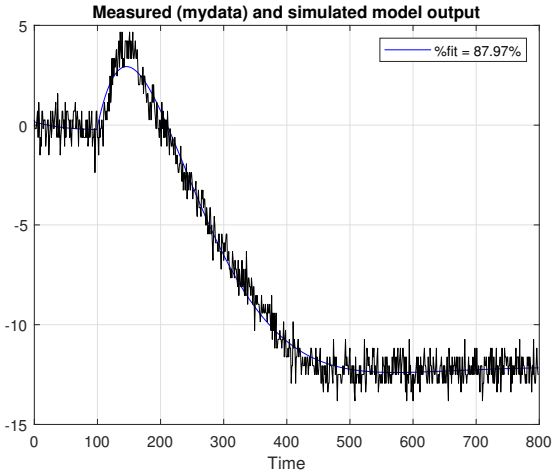


Figure 5: Comparing input-output model with data

2.2 Compressor speed step response (Disturbance)

For valve position step, the compressor speed is changed from 100 Hz to 150 Hz.The change in compressor speed is plotted with the change in θ.

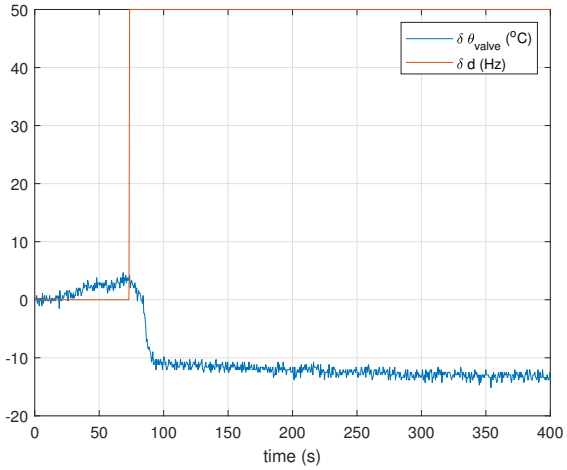


Figure 6: Disturbance Step Response

The following transfer function model is identified using MATLAB's system identification toolbox.

$$-G_d(s) = \frac{-0.006188}{s + 0.02448}$$

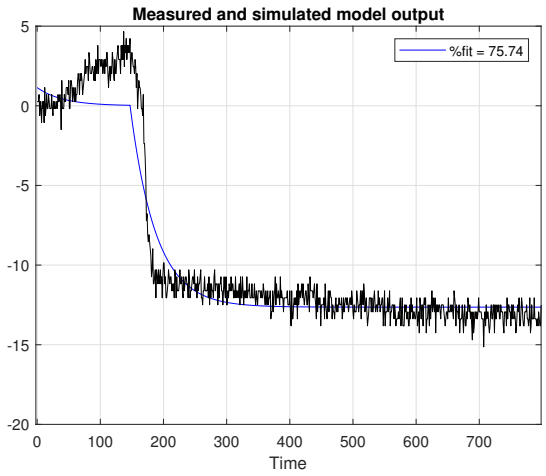


Figure 7: Comparing input-output model with data

3 Control Design and Demonstration

From the above data, the disturbance input bandwith is greater than the bandwith of the plant and there is a limitation on closed-loop disturbance disturbance rejection bandwith due to zero of the system.

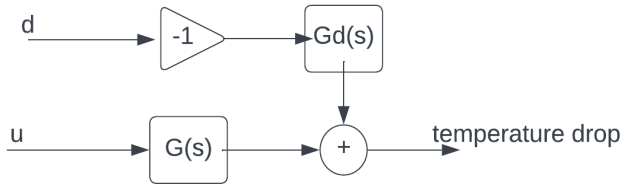


Figure 8: Block Diagram of the system

3.1 Controller Design

A PI controller is designed to improve the tracking performance. The disturbance rejection wich is limited by the zero of the plant is shown in the frequency response plots. The frequency correspinding to zero is chosen as the intergrator break frequency and the gain of the system is obtained using root-locus method.

$$PI\ controller: K\left(1+\frac{\omega_I}{s}\right)$$

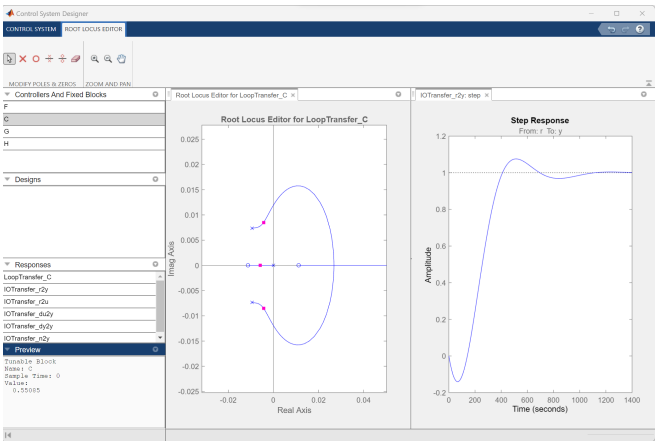


Figure 9: root locus method

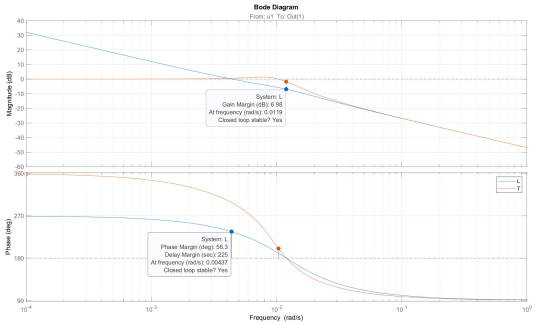


Figure 10: Frequency response

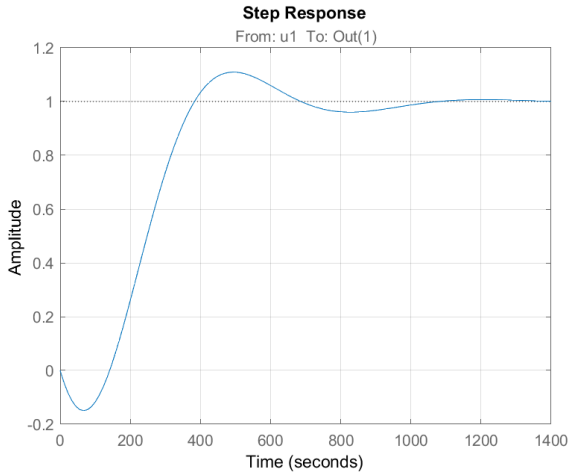


Figure 11: Step response

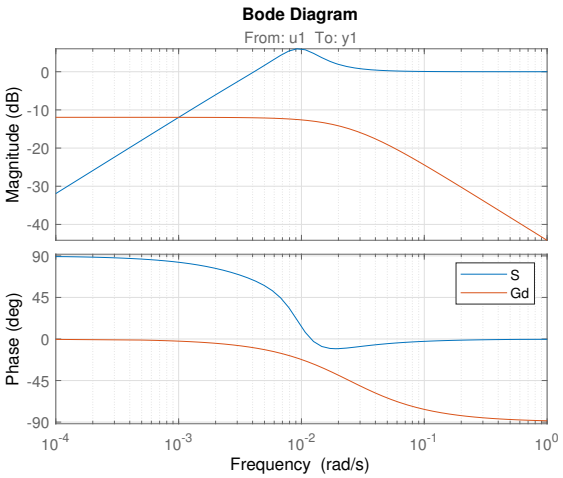


Figure 12: Sensitivity function

We have,

$$K = 0.58041 \qquad \omega_I = 0.0112$$

3.2 Controller Demonstration

The controller performance is demostrated in the case of maintaining the TC3 temperature at $-8\ ^oC$. The performance is shown in the following plots:

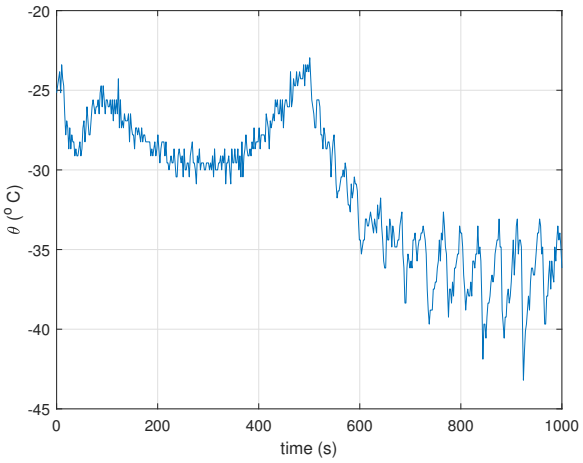


Figure 13: θ

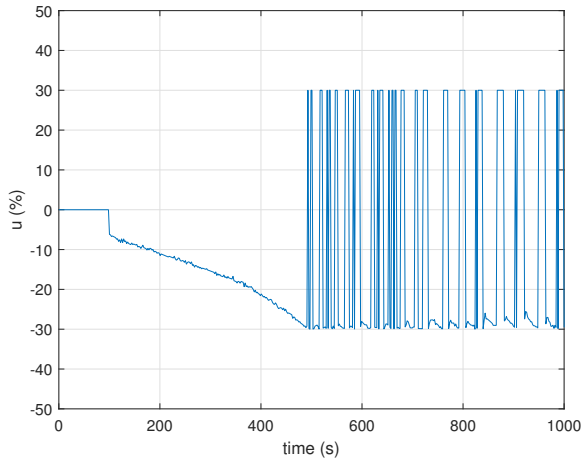


Figure 14: u (% valve opening)

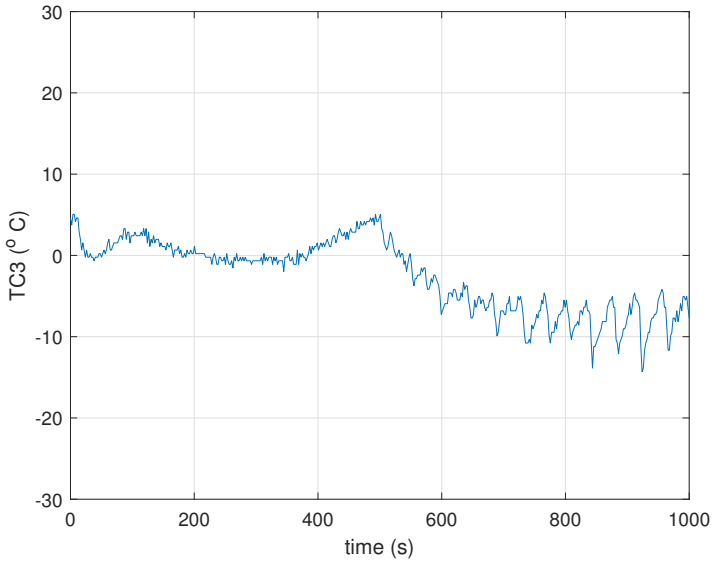


Figure 15: TC3

Conclusion

Clearly, the controller is oscillating between upper and lower saturation limits when the temperature is close to the setpoint. Further tunnning is necessary to correct for that behaviour. The system resoponse is extremely slow and the non-minimum phase nature of the resoponse makes it very non-intuitive to judge the performance in the early stages of the resoponse to tune it. The disturbance rejection performance is not satifactory.

Appendix

Code

Listing 1: Contoller code

```
/**
  ******************************************************************************
  * @file main.c
  * @author Zhou Zeng
  * @version V1.1.0
  * @date 10-Oct-2018
  * @brief Main program body.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ME586.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define MAX_COUNT 500 // max 500
#define TC2 4
#define TC3 3
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
float sensor_array[MAX_COUNT][3];
short time_count = 0;
short time_period = 500;
short jump = 0;
short jump_collect = 0;
int valve_30 = 2593;
int valve_50 = 2772;
// PID gains
float Kp = 0.58041;
float Ki = 0.5 + 0.58041 * 0.0112;
//
float TC3_desired = 0; //deg;
float TC3_measured = 0;
float TC2_measured = 0;
float theta_desired = 0;
float theta_measured = 0;
float e = 0;
float e_int = 0;
float u = 0;

/* Private function prototypes -----------------------------------------------*/
float TC2_ADC2tmep(int);
float TC3_ADC2tmep(int);
int val_ptoDAC(float);
float volt2temp(float);
int DAC_volt2count(float);
/* Private functions ---------------------------------------------------------*/


int main(void)
{
       initadc();
       initdac();
       initcom();
```

```
        inittime(time_period);

        disable_timer_interrupt();
        printf("KeyPress␣to␣start...␣\n\r");
        WaitForKeypress();

        restore_timer_interrupt();
        printf("Running␣acquisition...\n\r");

        while(time_count < MAX_COUNT){}

        printf("Printing...\n\r");
        printf("TC3_meas␣\t␣TC2_meas␣\t␣u\n\n\r");
        int i = 0;
        int j = 0;
        for (j = 0; j < MAX_COUNT; j++){
            for (i=0; i<=2; i++){
                        printf("%f", sensor_array[j][i]);
                        putchar('\t');
                }
            printf("\n\r");
        };
} //end of main program


void timehand(void){
    if (jump == 1){
            jump = 0;
            printf("\n\r");
            shownum(time_count);
            printf("\n\r");
            printf("TC3␣\t␣TC2␣\t␣e␣\t␣u\n\r");
            printf("%f", TC3_measured); putchar('\t');
            printf("%f", TC2_measured); putchar('\t');
            printf("%f", e); putchar('\t');
            printf("%f", u); putchar('\t');
            printf("\n\r");
    }
    else{
            // Control input
            TC2_measured = TC2_ADC2tmep(a_to_d(TC2));
            TC3_measured = TC3_ADC2tmep(a_to_d(TC3));
            if (time_count < 2){ //MAX_COUNT/10
                   d_to_a(0, val_ptoDAC(50));
                   }
            else{
                   // Data acquisition
                   theta_measured = TC3_measured - TC2_measured;
                   theta_desired = TC3_desired - TC2_measured;

                   //errors
                   e = theta_desired - theta_measured;
                   e_int = e_int + e*1;

                   // control input
                   u = Kp * e + Ki * e_int;
                   // saturation and anti-windup
                   if (u > 30){
                          u = 30;
                          e_int = 0;
                          }
```

```
                    else if(u < -30){
                           u = 30;
                           }
                    // output
                    d_to_a(0, val_ptoDAC(50+u));
                    }
        if (jump_collect == 1){
                // data acquisition
                sensor_array[time_count][0] = TC3_measured;
                sensor_array[time_count][1] = TC2_measured;
                sensor_array[time_count][2] = u;
                time_count += 1;
                if (time_count >= MAX_COUNT) disable_timer_interrupt();
                jump_collect = 0;
                }
        else{
                jump_collect = 1; // sample recording rate 2 sec. Sampling rate 1 sec.
                }
        jump = 1;
     }
}

// functions
float TC2_ADC2tmep(int count){
    float volt = 0.0022*count - 5.0047;
    return volt2temp(volt)+10;
}
float TC3_ADC2tmep(int count){
    float volt = 0.0022*count - 5.0179;
    return volt2temp(volt)+13;
}
int val_ptoDAC(float u){
    float volts = (u/100) * 4.096;
    return DAC_volt2count(volts);
}

float volt2temp(float volts){
        float temp = 200*volts - 250;
        return temp;
}

int DAC_volt2count(float volt){
    int counts = (int) (219.1978*volt + 2.3239e+03);
    return counts;
}

void inthand(void){
}

/**********************END OF FILE****/
```

Listing 2: Static Calibration

```
/**
  ******************************************************************************
  * @file main.c
  * @author Zhou Zeng
  * @version V1.1.0
  * @date 10-Oct-2018
  * @brief Main program body.
```

```
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ME586.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
int time_counts = 0;

int main(void)
{
        int valve[7] = {2503, 2593, 2683, 2772, 2862, 2952, 3042};
        int sensor_array[6] = {0, 0, 0, 0, 0, 0};
        int i = 0;
        int j = 0;

        initadc();
        initdac();
        initcom();
        inittime(100);

        while(1){
            disable_timer_interrupt();
            time_counts = 0;
            printf("Keypress␣for␣next␣input...\n\r");
            WaitForKeypress();
            restore_timer_interrupt();
            d_to_a(0, valve[j]);
            while(time_counts < 300){};
            printf("KeyPress␣to␣acquire␣...\n\r");
            WaitForKeypress();
            printf("Valve␣\t␣TC1␣\t␣TC2␣\t␣TC3␣\t␣TC4␣\t␣PR1␣\t␣PR2␣\n\r");
            shownum(valve[j]);
            putchar('\t');
            for (i=0; i<=5; i++){
                    sensor_array[i] = a_to_d(i);
                    shownum(sensor_array[i]);
                    putchar('\t');
            }
            printf("\n\r");
            j = j+1;
        };
} //end of main program


void timehand(void){
    time_counts = time_counts + 1;
    if (time_counts > 300){
        disable_timer_interrupt();}
}

void inthand(void){
}

/**********************END OF FILE****/
```

Listing 3: DAC Calibration

```
/**
  ******************************************************************************
  * @file main.c
  * @author Sesha charla & Mateus Bertolin
                          Zhou Zeng
  * @version V1.1.0
  * @date 18-Oct-2020
  * @brief AtoD and DtoA code.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ME586.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define DIGITAL_OUT 0
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/


int main(void)
{
        int value=2300;
        int adc = 0;
        initdac();
        initadc();
        initcom();
        printf("DAC␣\t␣ADC␣\n\r");
        while(1){
                WaitForKeypress();
                d_to_a(DIGITAL_OUT, value);
                WaitForKeypress();
                shownum(value);
                putchar('\t');
                WaitForKeypress();
                adc = a_to_d(4);
                shownum(adc);
                printf("\n\r");
                value += 100;
                if (value>=3600) {
                    break;
               };
        };
} //end of main program


void timehand(void){
}

void inthand(void){
}

/**********************END OF FILE****/
```

Listing 4: Step Input

```c
/**
  ******************************************************************************
  * @file main.c
  * @author Zhou Zeng
  * @version V1.1.0
  * @date 10-Oct-2018
  * @brief Main program body.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ME586.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define MAX_COUNT 800
#define TC2 4
#define TC3 3
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
int sensor_array[MAX_COUNT][2];
short time_count = 0;
short time_period = 500;
short jump = 0;
int valve_30 = 2593;
int valve_50 = 2772;
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/


int main(void)
{
	initadc();
	initdac();
	initcom();
	inittime(time_period);

	disable_timer_interrupt();
	printf("KeyPress to start... \n\r");
	WaitForKeypress();

	restore_timer_interrupt();
	printf("Running acquisition...\n\r");

	while(time_count < MAX_COUNT){}

	printf("Printing...\n\r");
	printf("TC2 \t TC3\n\r");
	int i = 0;
	int j = 0;
	for (j = 0; j < MAX_COUNT; j++){
	    for (i=0; i<=1; i++){
	                shownum(sensor_array[j][i]);
	                putchar('\t');
	        }
	    printf("\n\r");
	};
} //end of main program


void timehand(void){
```

```
    if (jump == 1){
            jump = 0;
    }
    else{
            // Control input
            if (time_count < 100){
                d_to_a(0, valve_50);
            }
            else{
                d_to_a(0, valve_30);
            }

            // Data acquisition
            sensor_array[time_count][0] = a_to_d(TC2);
            sensor_array[time_count][1] = a_to_d(TC3);

            time_count += 1;
            if (time_count >= MAX_COUNT) disable_timer_interrupt();
            jump = 1;
        }
}

void inthand(void){
}

/**********************END OF FILE****/
```

Listing 5: Disturbance Step Input

```
/**
  ******************************************************************************
  * @file main.c
  * @author Zhou Zeng
  * @version V1.1.0
  * @date 10-Oct-2018
  * @brief Main program body.
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "ME586.h"

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define MAX_COUNT 800
#define TC2 4
#define TC3 3
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/
int sensor_array[MAX_COUNT][2];
short time_count = 0;
short time_period = 500;
short jump = 0;
int valve_30 = 2593;
int valve_50 = 2772;
int valve_40 = 2683;
/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/


```

```
int main(void)
{
        initadc();
        initdac();
        initcom();
        inittime(time_period);

        disable_timer_interrupt();
        printf("KeyPress to start...\n\r");
        WaitForKeypress();

        restore_timer_interrupt();
        printf("Running acquisition...\n\r");

        while(time_count < MAX_COUNT){}

        printf("Printing...\n\r");
        printf("TC2 \t TC3\n\r");
        int i = 0;
        int j = 0;
        for (j = 0; j < MAX_COUNT; j++){
            for (i=0; i<=1; i++){
                        shownum(sensor_array[j][i]);
                        putchar('\t');
                }
            printf("\n\r");
        };
} //end of main program


void timehand(void){
    if (jump == 1){
            jump = 0;
    }
    else{
            // Control input
            if (time_count < 100){
                d_to_a(0, valve_40);
            }
            else{
                d_to_a(0, valve_40);
            }

            // Data acquisition
            sensor_array[time_count][0] = a_to_d(TC2);
            sensor_array[time_count][1] = a_to_d(TC3);

            time_count += 1;
            if (time_count >= MAX_COUNT) disable_timer_interrupt();
            jump = 1;
        }
}

void inthand(void){
}

/**********************END OF FILE****/
```

Listing 6: Input/Output Identification

```
clear; clc;
load("step_data.mat");

TC2_counts = InputStep(:, 1);
TC3_counts = InputStep(:, 2);

TC2 = volt2temp(TC2_count2volt(TC2_counts));
TC3 = volt2temp(TC3_count2volt(TC3_counts));

t = linspace(0, 800*(0.5), 800);
theta = TC3 - TC2;
theta = theta - mean(theta(1:10,1));


% start of step 148
u = ones(800, 1);
u(1:99, 1) = u(1:99, 1) * 0;
u(100:end, 1) = u(100:end, 1) * -20;


figure(1)
plot(t, theta);
hold on;
plot(t, u);
xlabel("time (s)")
grid on
legend("\delta \theta_{valve} (^oC)", "\delta u (%)")
```

Listing 7: Disturbance Identification

```
clear; clc;
load("dist_data.mat");

TC2_counts = DisturbanceStep(:, 1);
TC3_counts = DisturbanceStep(:, 2);

TC2 = volt2temp(TC2_count2volt(TC2_counts));
TC3 = volt2temp(TC3_count2volt(TC3_counts));

t = linspace(0, 800*(0.5), 800);
theta = TC3 - TC2;
theta = theta - mean(theta(1:10,1));


% start of step 148
u = ones(800, 1);
u(1:147, 1) = u(1:147, 1) * 0;
u(148:end, 1) = u(148:end, 1) * 50;


figure(1)
plot(t, theta);
hold on;
plot(t, u);
xlabel("time (s)")
grid on
legend("\delta \theta_{valve} (^oC)", "\delta d (Hz)")
```

Listing 8: volt2temp()

```
function temp = volt2temp(volts)
temp = (volts-1.25)/0.005;
end
```

Listing 9: TC2_count2volt()

```
function volt = TC2_count2volt(count)
% coefficients 0.0022 -5.0047
volt = 0.0022*count - 5.0047;
end
```

Listing 10: TC3_count2volt()

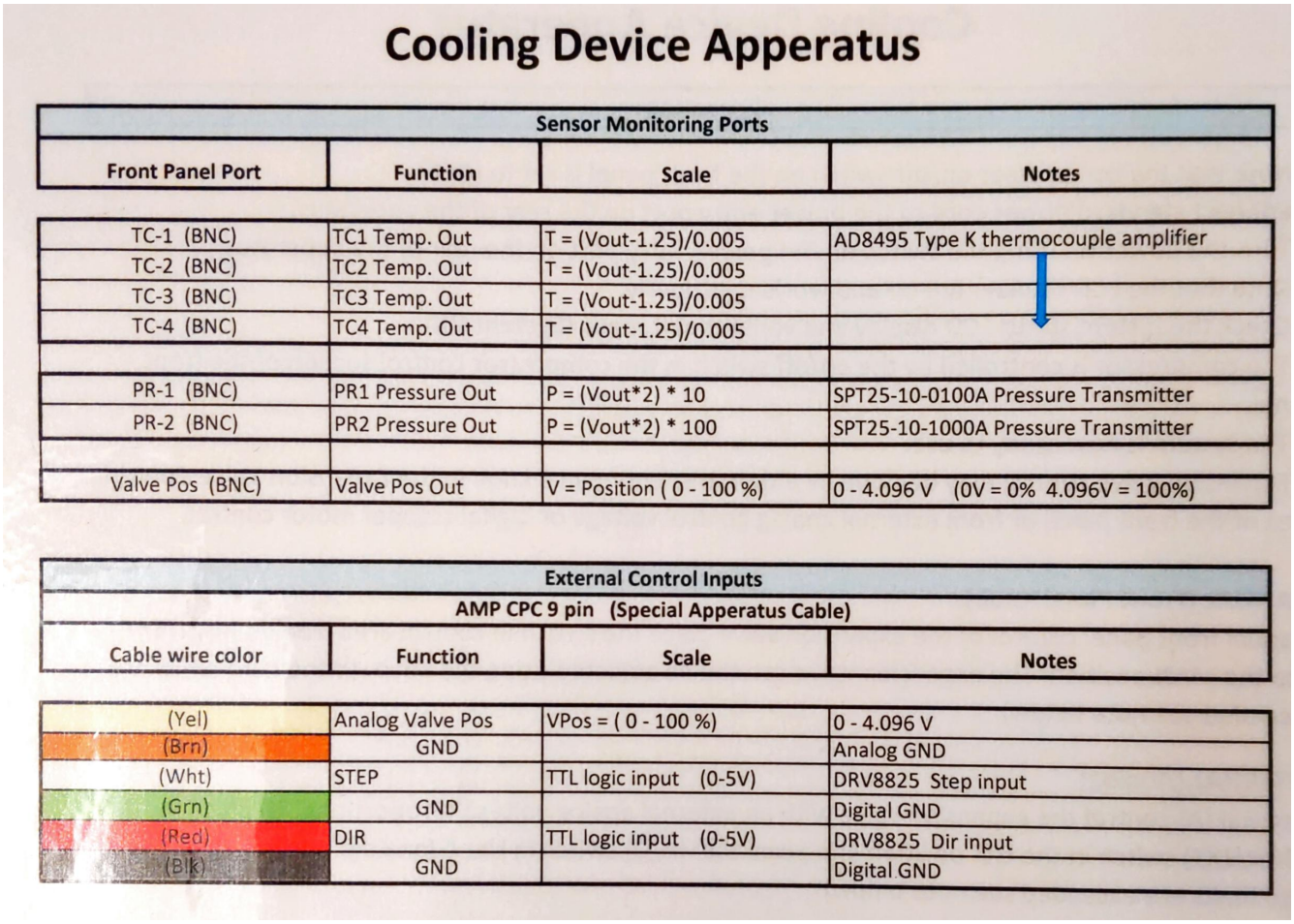
```
function volt = TC3_count2volt(count)
% coefficients 0.0022 -5.0179
volt = 0.0022*count - 5.0179;
end
```

Listing 11: TC3_count2volt()

```
    clear; clc;
load("dist_tf.mat");
load("input_tf.mat");
G = input_tf;
Gd = -dist_tf; %tf(-dist_tf.Numerator, dist_tf.Denominator);
%% Disturbance input parameters
kd = Gd.Numerator;
wd = Gd.Denominator(1, 2);
%% Plant parameters
% numerator dynamics
b0 = G.Numerator(1, 1);
b1 = G.Numerator(1, 2);
wz = -b1/b0;
% denominator dynamics
a1 = G.Denominator(1, 2);
a2 = G.Denominator(1, 3);
wn = sqrt(a2);
zeta = a1/(2*wn);
%%
wI = wz;
K = 0.58041;
LD = (wz/wd)*tf([1, wd],[1 wz]);
C = K*tf([1, wI], [1, 0]);
L = C*G;
%rltool(L)
%%
T = feedback(L, 1);
S = 1/(L+1);
figure()
bode(L, T)
grid on;
legend("L", "T")
figure()
step(T)
grid on;
figure()
bode(S, Gd)
```

```
grid on;
legend("S", "Gd")
```

Lab Manual



Cooling Device Apperatus

Sensor Monitoring Ports			
Front Panel Port	**Function**	**Scale**	**Notes**
TC-1 (BNC)	TC1 Temp. Out	T = (Vout-1.25)/0.005	AD8495 Type K thermocouple amplifier
TC-2 (BNC)	TC2 Temp. Out	T = (Vout-1.25)/0.005	
TC-3 (BNC)	TC3 Temp. Out	T = (Vout-1.25)/0.005	
TC-4 (BNC)	TC4 Temp. Out	T = (Vout-1.25)/0.005	
PR-1 (BNC)	PR1 Pressure Out	P = (Vout*2) * 10	SPT25-10-0100A Pressure Transmitter
PR-2 (BNC)	PR2 Pressure Out	P = (Vout*2) * 100	SPT25-10-1000A Pressure Transmitter
Valve Pos (BNC)	Valve Pos Out	V = Position (0 - 100 %)	0 - 4.096 V (0V = 0% 4.096V = 100%)

External Control Inputs AMP CPC 9 pin (Special Apperatus Cable)			
Cable wire color	**Function**	**Scale**	**Notes**
(Yel)	Analog Valve Pos	VPos = (0 - 100 %)	0 - 4.096 V
(Brn)	GND		Analog GND
(Wht)	STEP	TTL logic input (0-5V)	DRV8825 Step input
(Grn)	GND		Digital GND
(Red)	DIR	TTL logic input (0-5V)	DRV8825 Dir input
(Blk)	GND		Digital GND

Figure 16: Sensor calibration equations

Purdue University
School of Mechanical Engineering

ME 586: Microprocessors in Electromechanical Systems — **Fall Semester 2022** — **Dr. P. Meckl**

Project Assignment on Adaptive Refrigeration Cycle (Tentative)

Objectives:

1. To Study the Dynamics of a Refrigeration Cycle.
2. To Study the Effects of a Capillary Tube (Expander).
3. To Use Simulink to Model a Control System.
4. To Investigate Adaptation of a Refrigeration Cycle using a Variable Expansion Valve.

Procedure:

A refrigeration cycle is composed of four main components. These are: compressor, condenser, capillary tube (expander), and evaporator. The role of the capillary tube is very important in the overall operation and thermal efficiency of the cycle. When high pressure saturated liquid leaves the condenser, it passes through the capillary tube, which drops the pressure and delivers a low-pressure two phase refrigerant to the evaporator. From a modeling point of view, the capillary tube is a resistive element in both the thermal and fluid domains. In this experiment, we are interested in studying the effect of the capillary tube on the dynamics of the refrigeration cycle.

Shown on the next page is a schematic of the experimental setup. It consists of three of the four main components of the refrigeration cycle, and the fourth element, the capillary tube, is replaced by an expansion valve, which is driven by a stepper motor. There are two pressure transducers located at the low and high pressure sides of the compressor. There are four thermocouples located at the inlets and outlets of the condenser and evaporator.

The stepper motor is driven by the pulse train generated by the microcontroller and sequenced properly by the translator card. Two signals should be sent by the microcontroller: one to identify direction of the valve (opening or closing), the other for driving the motor (pulse train).

The four thermocouple output signals have to be compensated for zero degree base and amplified before being sent to the microcontroller. This is achieved by sending the thermocouple outputs through the circuits shown on page 3. As shown, signals from pressure transducers are also amplified by using an op-amp. The result of this signal conditioning is such that for the thermocouples we have 125 mV/°C and for pressure transducers we have 67 mV/psi conversion factors.

For the expansion valve motor assembly, a high logical signal for motor direction will turn the valve clockwise, i.e., will open the valve. Also it takes 3800 pulses of the stepper motor to go from a full-open to a full-closed position of the valve.

Note that *fast opening of the valve may damage the compressor*, because it results in a surge of liquid coolant.

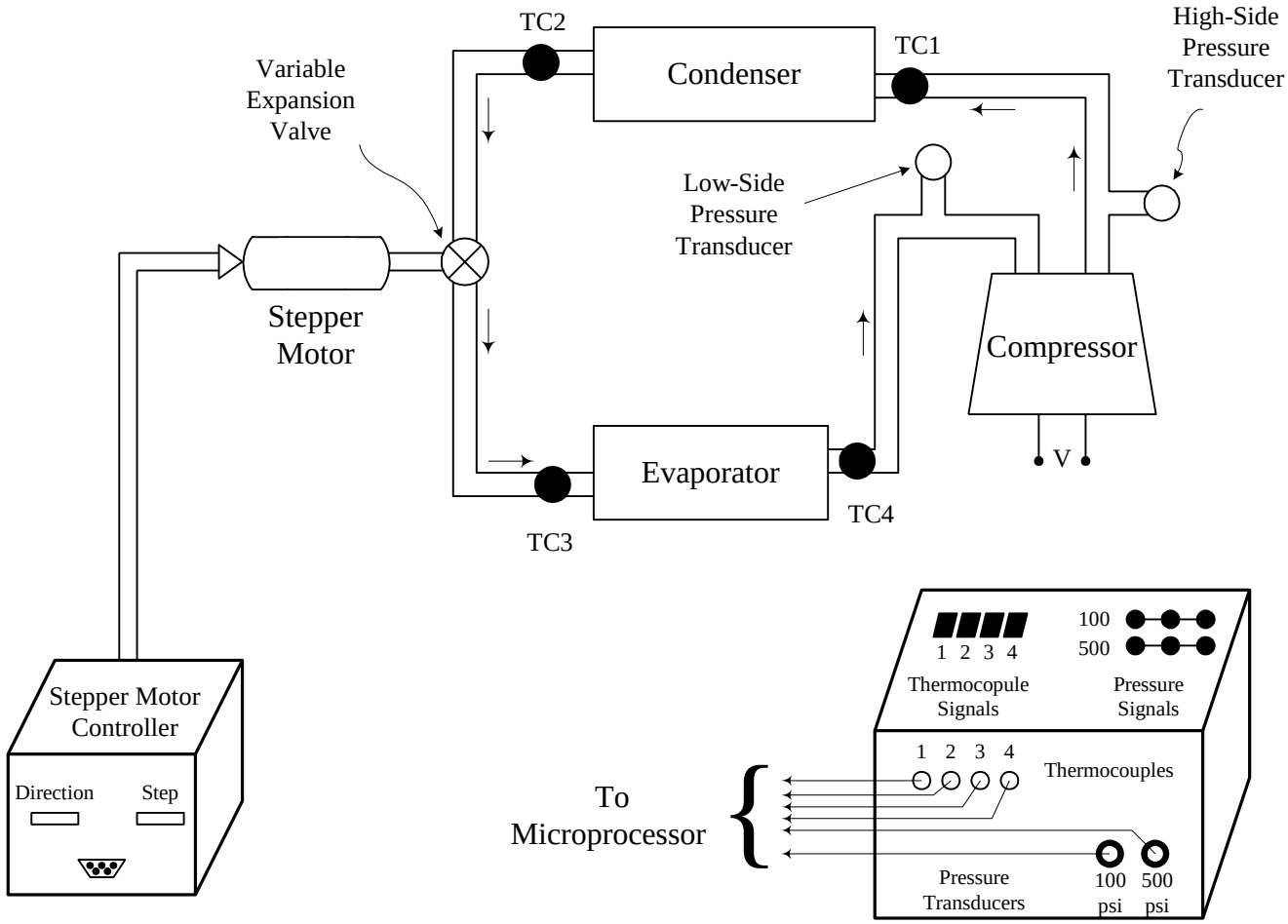


Figure 1: Refrigeration System Schematic

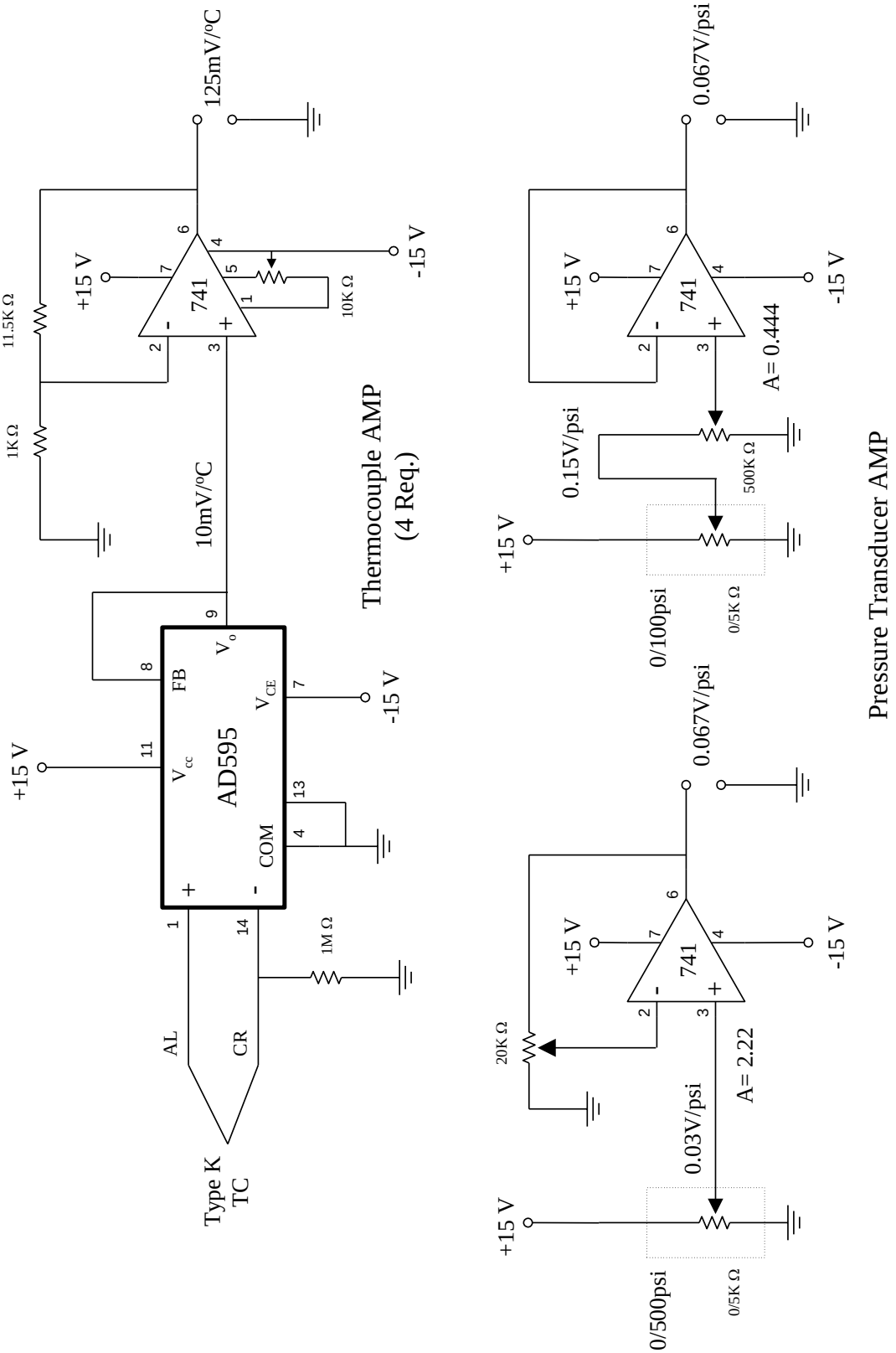


Figure 2: Internal Wiring of Thermocouple box for Signal Conditioning

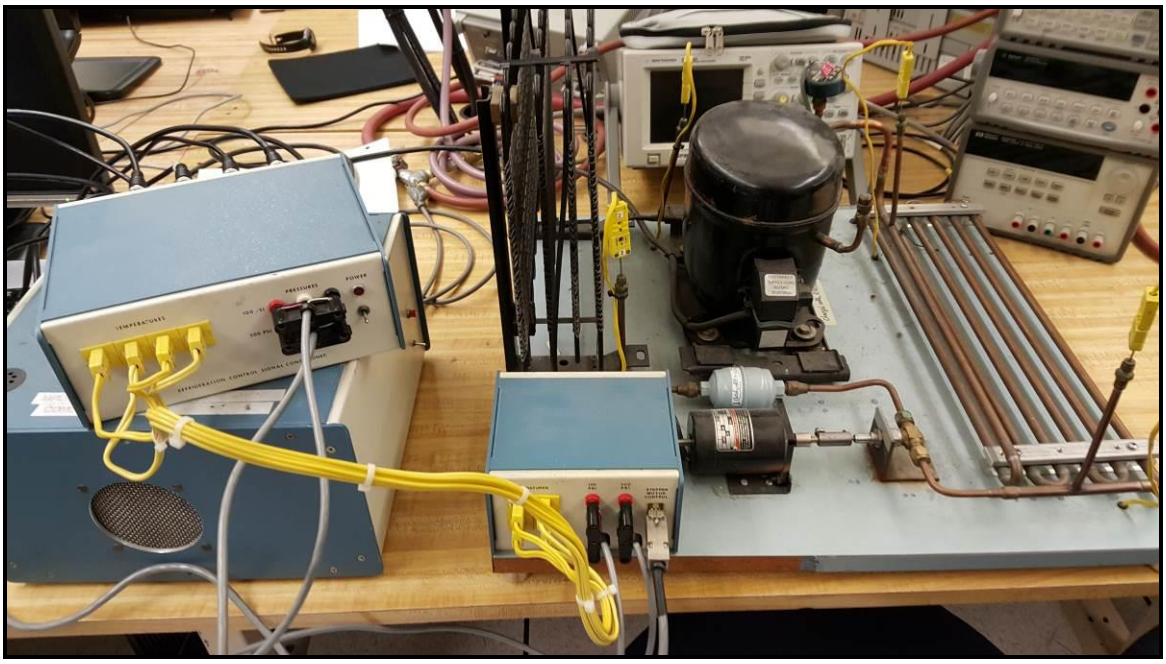


Figure 3: Refrigeration System

Since the thermocouple and pressure transducer signals need to be conditioned before sending them to the microcontroller, the signals will be fed into the Thermo-Couple Box as shown in Figures 4 and 5.

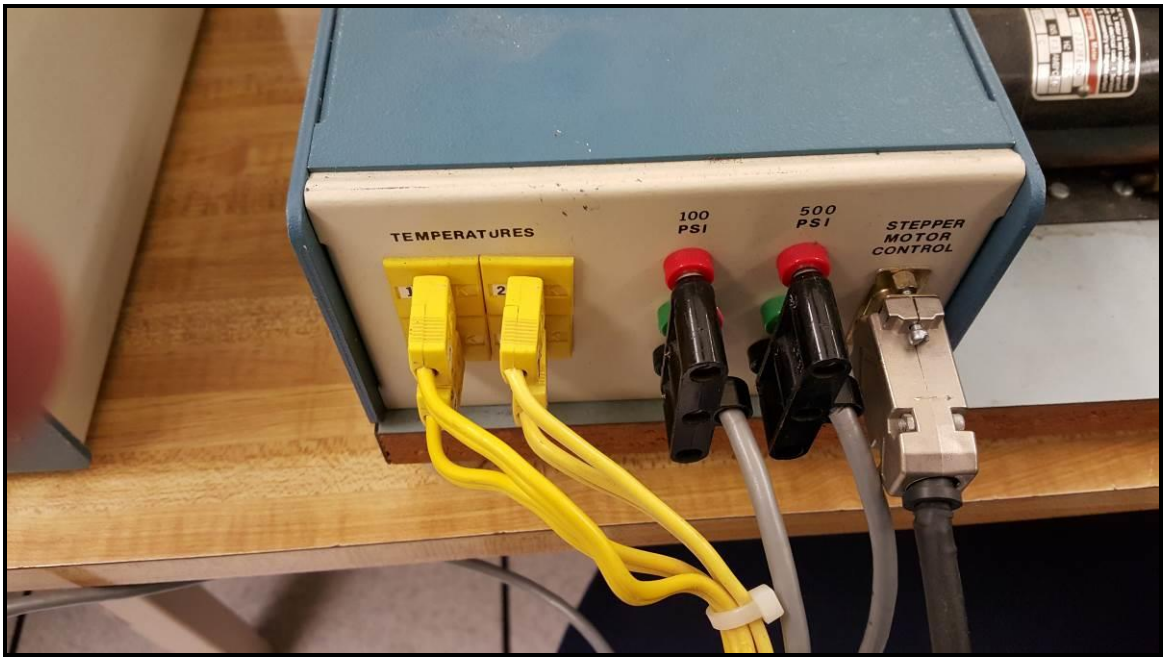


Figure 4: Signal Connections on Experimental Apparatus

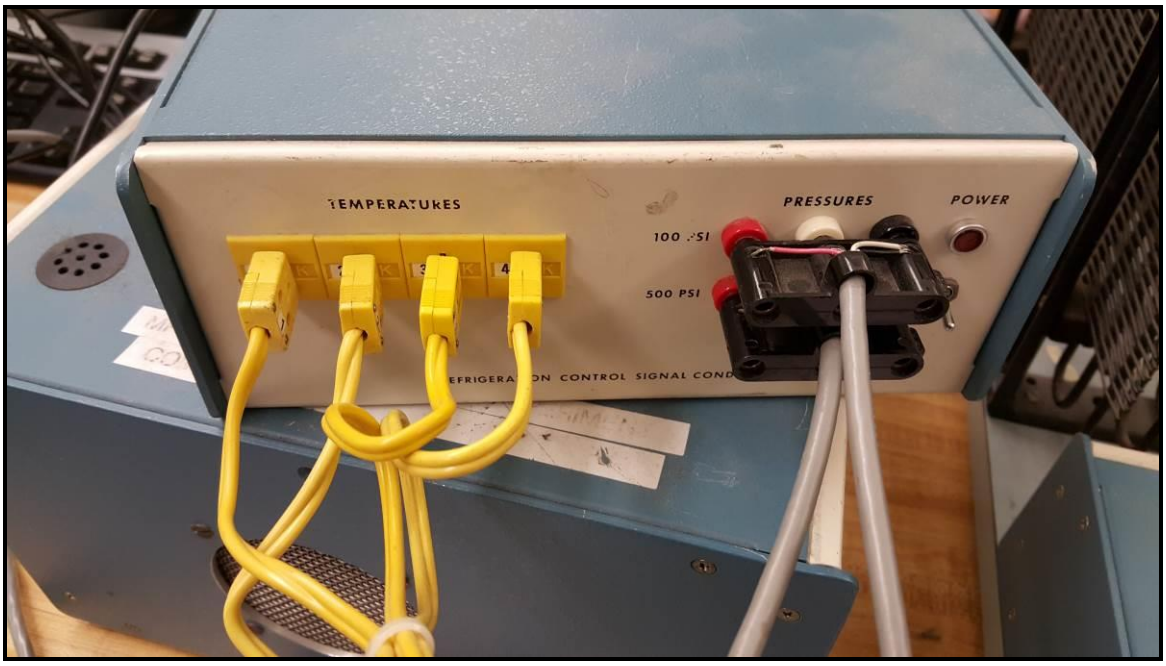


Figure 5: Signal Connections on Thermocouple Box

The back of the Thermocouple box has BNC connections that can be fed into the ADC ports of the STM32, as shown in Figure 6.

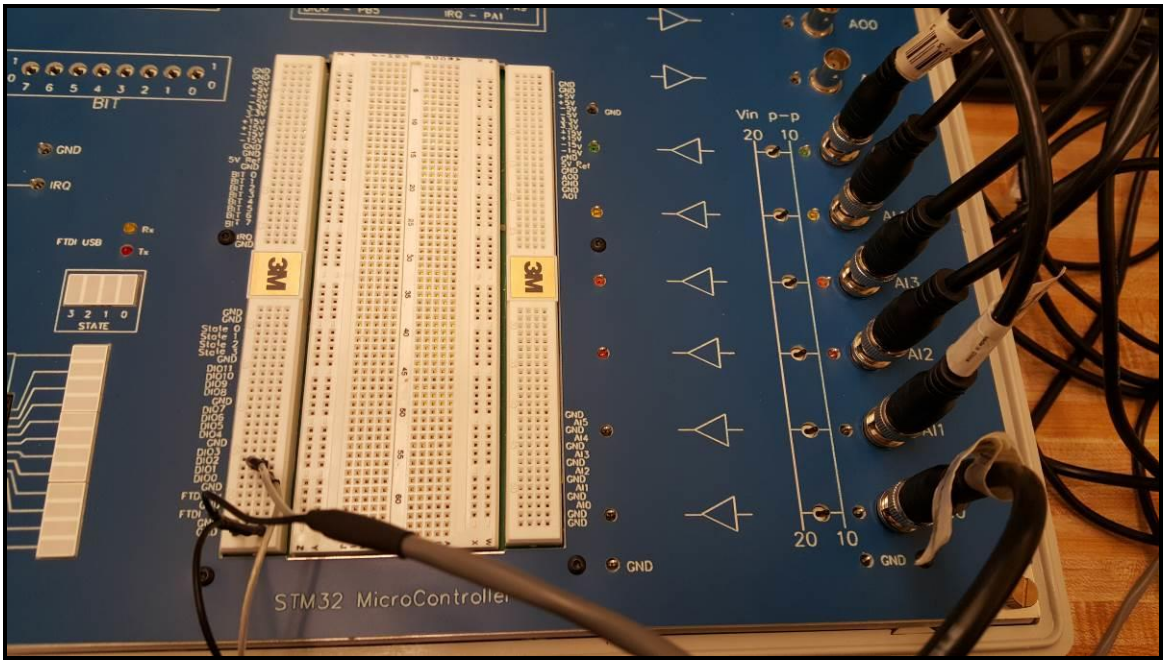


Figure 6: Signal Connections on Thermocouple Box

In order to control the stepper motor you need generate a pulse train. This will then be fed into the Stepper Motor Controller box as shown below in Fig. 7.

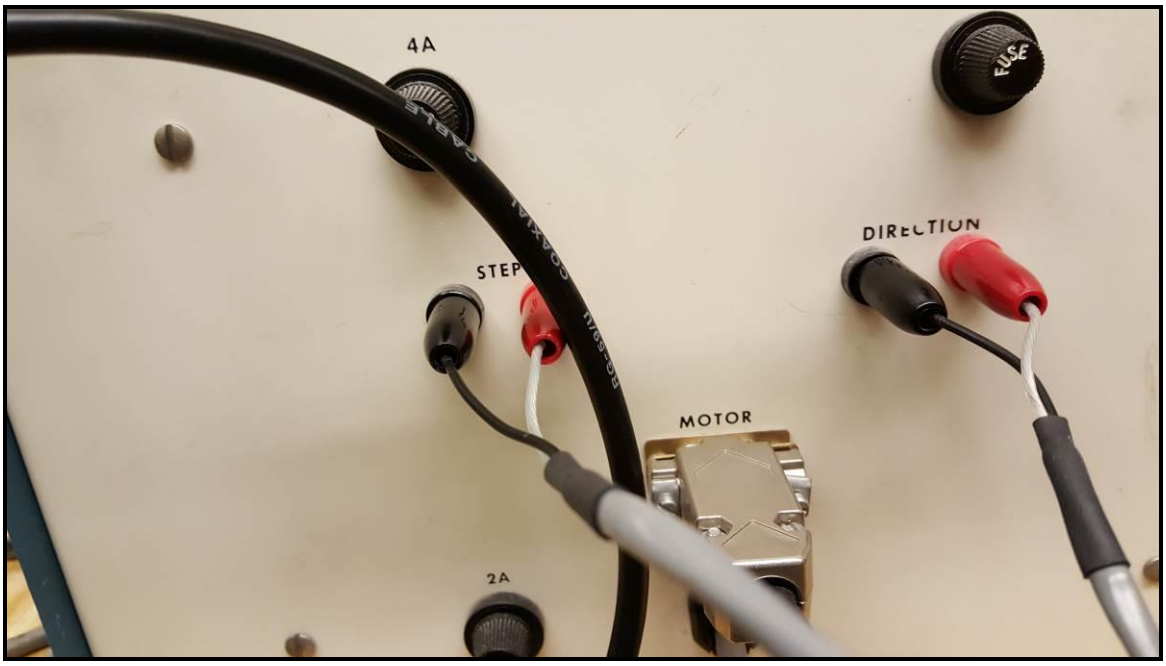


Figure 7: Stepper Motor Controller Connections

Note that both Step and Direction are digital signals. You can see in Figure 6 (lower left corner) that they are connected to digital pins on Port B that can send out these signals

Tasks:

1. Write a program to open and close the expansion valve. Implement this program on the system with the compressor off. How would you change the rate of opening and closing of the valve?

2. Keep the expansion valve in the midway open position. Turn the compressor ON, and check the system charge by reading the high side pressure. Change the position of the expansion valve and record the thermocouple reading at each position.

3. Can you derive a relationship between the expansion valve opening and the outlet temperature of the evaporator and inlet temperature of the condenser?

4. Two of the important temperatures in the cycle are the inlet temperature to the condenser and the outlet temperature of the evaporator. Based on your readings of part (2), pick a desired temperature for the evaporator and/or condenser. Design a controller that adjusts the expansion valve so that these temperatures will always remain at the desired levels regardless of any disturbances.

5. Develop a Simulink model of your system, including the refrigerator and controller. Verify that your controller performs as expected.

6. Implement your controller and study the results, by using a hot air blower over the evaporator and over the condenser. Use different settings for the blower. Make sure you also compare your experimental results to your Simulink responses.