UNIVERSITY OF VICTORIA
MECH 458 Mechatronics

Sorting Machine Product Design

Final Report

Names: Gucheng Wang V00942654
Cheuk Ki Liu V00937822

To: Xinxin Shang
Date: April 23 rd, 2022

List of Figures

Introduction

In this project, we are going to design a mechanical sorting system. The sorting system is expected to detect and classify four different kinds of materials by sensors and actuators. Those four materials include Aluminum, Steel, Black plastic and White plastic. To make the sorting system work properly, it is required the knowledge of circuit wiring, the functions of ADC (Analog-to-Digital Converter), different kinds of interrupts (rising edge and falling edge), PWM (Pulse Width Modulation), stepper motor, and DC motor. Furthermore, an embedded programming technique is required, including consideration of stepper motor hissing effects and sensor debouncing.

System Technical Documentation

System High-Level Diagram

This figure below shows the block diagram of the whole system. It includes some sensors and actuators, which correspond to the input and the output of the system. The detailed functions of the components, like ADC and the stepper motor, would be given in the flow chart part of the sections.

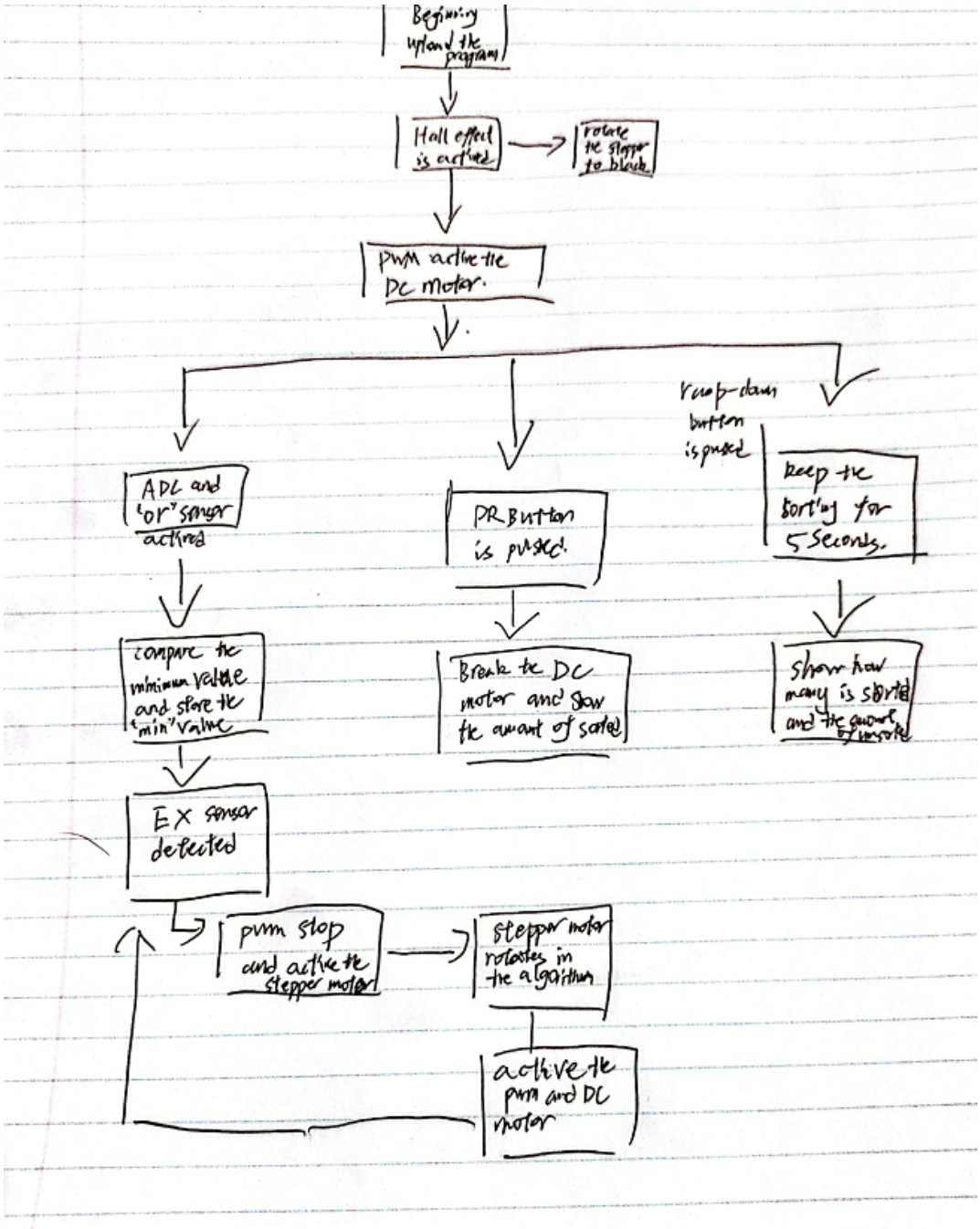

Figure 1: The block diagram of the whole system

Circuit Diagram

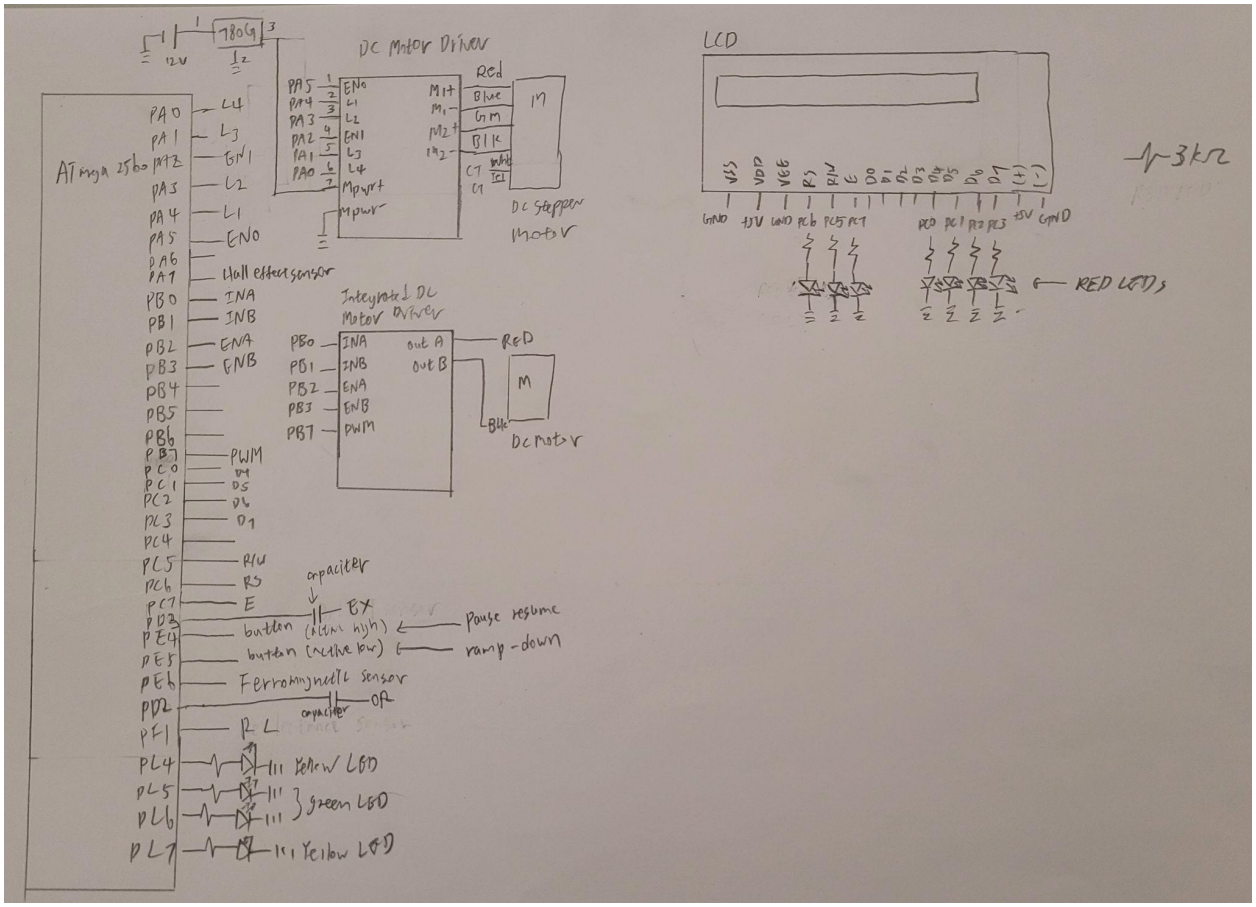

Figure 2: The circuit connection of the system

List of pins connection:

PA0	L4 (DC step motor driver)
PA1	L3 (DC step motor driver)
PA2	EN1 (DC step motor driver)
PA3	L2 (DC step motor driver)
PA4	L1 (DC step motor driver)
PA5	EN0 (DC step motor driver)
PA7	HE sensor
PB0	INA (Integrated DC motor Driver)
PB1	INB (Integrated DC motor Driver)

PB2	ENA (Integrated DC motor Driver)
PB3	ENB (Integrated DC motor Driver)
PB7	PWM
PC0 with RED LED & 3kΩ	D4 (LCD)
PC1 with RED LED & 3kΩ	D5 (LCD)
PC2 with RED LED & 3kΩ	D6 (LCD)
PC3 with RED LED & 3kΩ	D7 (LCD)
PC4	RED LED & 3kΩ
PC5 with RED LED & 3kΩ	R/W (LCD)
PC6 with RED LED & 3kΩ	RS (LCD)
PC7 with RED LED & 3kΩ	E (LCD)
PD2	OR sensor with capaciter
PD3	EX sensor with capaciter
PE4	Pause-resume buttom (active high)
PE5	Ramp-down button (active low)
PF1	RL sensor
PL4	Yellow LED with 1kΩ
PL5	Green LED with 1kΩ
PL6	Green LED with 1kΩ
PL7	Yellow LED with 1kΩ

Table 1: GPIOs connection

System Algorithm

The specific code written for this system is presented in Appendix A. The algorithm will be explained here.

Algorithm of the main structure:
Firstly, I would talk about the main structure of our algorithm. To begin with, since we are designing an embedded program. The first thing done in the main program is the initialization. The initialization includes all the hardware settings, like the system clock, all the interrupts which are OR sensor, Ex sensor, Pause button and Ramp-down button) and the configuration of ADC. After that, the program would go into a “while loop” statement. In the while loop statement, the system would keep checking three flags to monitor if the Pause/Resume button and the ramp down button is pressed. Or the adc_flag is triggered and the system will compare the “min” value of the ADC value to adjust what kind of objects are scanned.

Data structure for storing elements:
For our data stored structure, we choose to use “array” to store the “min” value which stands for different kinds of objects. After one “min” value is achieved in the period of the “OR” sensor, it will be added to the next position of the array, and there is no limit for the amount of array. This progress will happen in the ADC interrupt.

Stepper motor:
After the “min” value is stored in the array, it could be used to calculate the rotate_check number which is used to decide how the stepper motor will rotate. We set two parameters which are “curpos” and “furpos” and rotate_check will be their difference. In this case, both “curpos” and “furpos” will be the value stored in array, the “curpos” is the current position and “furpos” is the future position. And there are 4 numbers for positions, which are 0,1,2,3, representing black, aluminum, white and steel, and when the difference between two positions is 0, the stepper motor will not move; when the difference is -2, and 2, the stepper motor will rotate 180 degree; when the difference is 1 and -3, the stepper motor will rotate 90 degree clockwise, and when the difference is -1 and 3, the stepper motor will rotate 90 degree counterclockwise. And this calculation will happen in the EX interrupt, after the EX sensor detects the target, PWM will be stopped and the stepper motor will rotate as the calculation result.

Overall, the main logic of our program is that using array to store the “min” value measured by the “IR” and “OR” sensor and calculate the parameter rotate_check to decide how the stepper motor will rotate.

Flow chart

Stepper motor flowchart

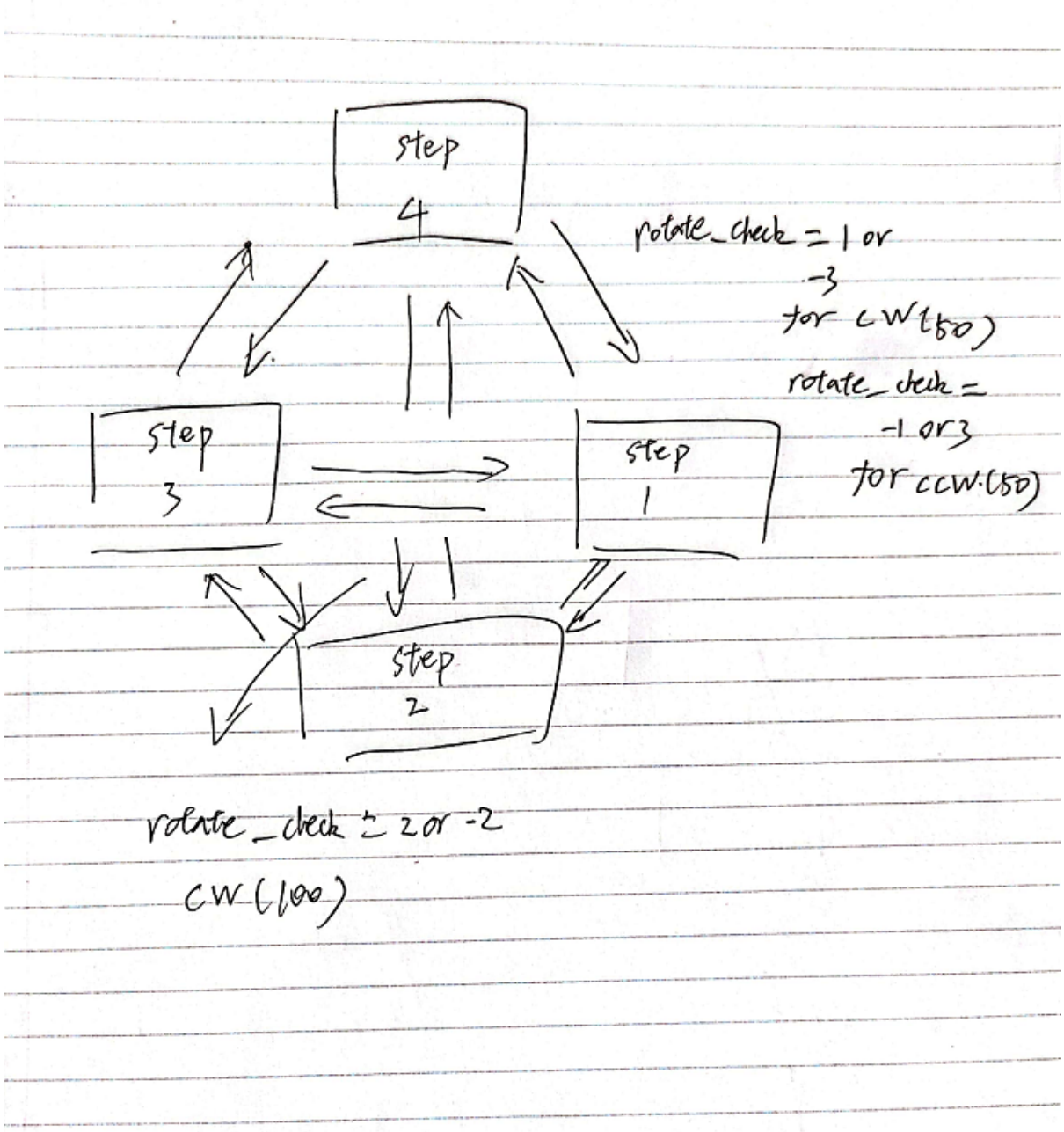

Figure 3: Finite state machine flowchart of the stepper motor

The flowchart of the stepper motor is similar to a finite state machine of four states. Each state corresponds to the first state of two-phase winding in the stepper motor. The step required for turning the motor depends on the previous position of the tray and the next position of the tray for correct sorting always turns clockwise 90° (50 steps), counter-clockwise 90°(50 steps), and 180°(100 steps).

ADC reading flowchart

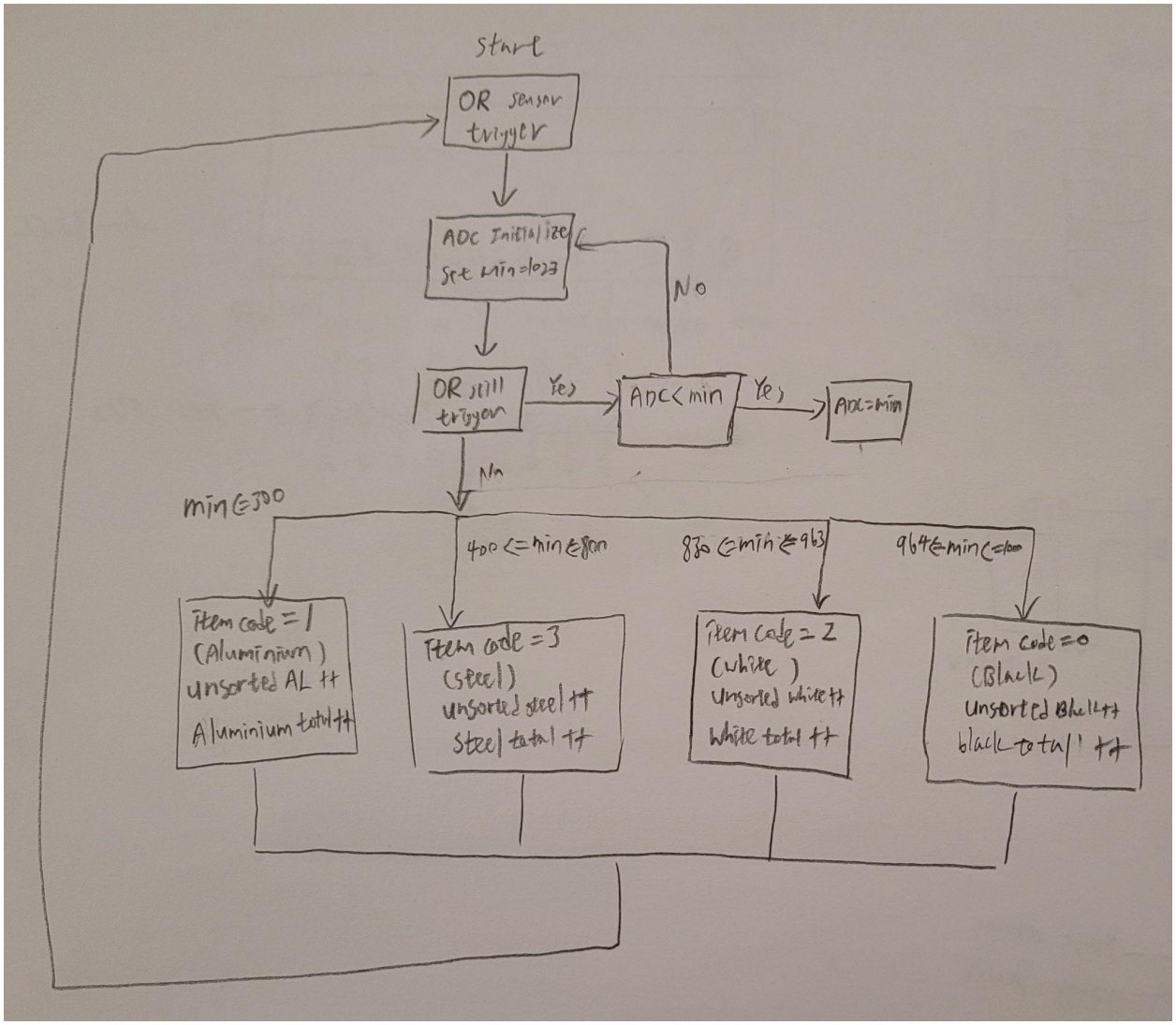

Figure 4: Flowchart for the ADC measurements

To make the system know the sorting materials, we need to obtain the minimum reflectiveness of the materials from ADC. After we get all the range of the ADC values corresponding to the specific materials, we have the variable “item code” for different materials. 0 for black plastic, 1 for aluminum, 2 for white plastic and 3 for steel.

System Performance Specification

There are 3 major parts to determine the system performance. The parameters include the PWM, ADC sensor reflectiveness, the acceleration-deceleration profile and DC motor phase windings.

Firstly, the PWM is a periodic square signal which controls the speed of the DC motor. If the duty cycle (in percentage) is large in a PWM period, the DC motor would have a high velocity. However, if the DC motor runs too fast, the inertia at the end of the conveyor belt would be too large. As a result, the object may fall into the wrong sorting tray. This requires coordination between both the stepper motor and DC motor. Based on our experiment, when we set the duty cycle to 125/255, the classification correct rates decrease. When we choose the duty cycle to be 85/255, the classification of materials is safe. The reason is lower speed can prevent the system from being interrupted by the sensors and lower the inertia of the conveyor belt.

Secondly, ADC reflectiveness parameters which is the main part of identifying the materials. When the material passes through the OR sensor it triggers the IR sensor which connects to ADC, and the IR sensor will read through the object's surface area. However, the largest value in the data obtained from different materials might be the same, so we just want to store the minimum reads from the object by resetting the minimum value in the ADC.

The range of the material is listed below:

Aluminum: Aluminum <= 300
Steel: 400 <= Steel <= 800
White Plastic: 830 <= White <= 963
Black Plastic: 964 <= Black <= 1000

The last important parameter is the acceleration which determines the speed of the sorting system. In our project, we use the array to adjust the delay function in each rotation step. And we are using a two-phase stepper motor to increase the rotation speed. For example, when we want the stepper to turn 90 degrees which means it needs to turn 50 steps because each step rotates 1.8 degrees. To make the stepper accelerate, we decrease the delay time for each step, but we found that the delay time cannot be very small. Otherwise, the motor will not work.

Testing and Calibration Procedure

In order to make the testing and the calibration procedure efficient, we separate a few parts to test on different days.

1 st milestone:

- Make sure the hall effect sensor works properly to initialize the stepper motor
- Make sure the PWM is working to adjust the speed of the DC motor
- Make sure the ADC setting is correct and able to read all the materials
- Make sure the LCD can display the data we want from the code
- All the interrupts are configured in the correct way

2 nd milestone:

- Make an array to store the reading of materials
- Make sure the pause-resume button is implemented
- Make sure the sensors can communicate and corporate with each other

3 rd milestone:

- The synchronization of the whole project
- Ramp down implemented
- System can sort materials correctly
- Optimize the system (improve the speed and the accuracy of the sorting system)

The benefit for separate different tasks in the project is to minimize the error and fix error more efficiently. For example, before making the array to store the reading of materials in the 2nd milestone, we have to make sure the ADC is working, it can read and rest the minimum value every time so that when the other material has a higher value than the previous materials, the reading will not be lost. And to decide the range of materials reading, we have to make sure the LCD is displaying the materials read by the IR sensor.
To ensure the subsections of the system are accurate, there are two main debugging techniques, using LEDs and LCDs. For example, when we want to test if the EX sensor is implemented correctly, we can set the LED to light up when the EX sensor is triggered. On the other hand, the fastest way to test a sensor and the kind of sensor interruption, we can use a multimeter to measure the voltage between the sensors. If it changes from high (5V) to low (0V), it is a falling edge interrupt, otherwise, it is a rising edge interrupt.

Limitation and System Tradeoffs

One of our project limitations is the velocity of the DC motor, for the progress of measuring the objects happens in the “main”, thus if the DC motor moves in a high velocity, the ADC interrupt might come before the measuring “min” value progress ends. And for this reason, we set the speed of the DC motor to 0x55, which is very close to the limit, and if the speed goes further than this value, there will be mistakes.

And for the stepper motor acceleration, there are also some limits, in order to let the stepper motor rotate at a higher speed, we try to decrease the time delay of the timer. In our code, the lowest value of the time delay is 6 ms, and at that moment, the stepper motor will rotate at its highest speed. It makes limits the speed of the stepper motor, however, if we make the highest value of the time delay decrease for 1 to 4 ms, we might set the lowest value of time delay to 4ms. And it will give better results.

For the third limit, it is our highest amount of sorted objects. The array we set to store the “min” value could only store 100 numbers. If the objects` number goes beyond 100, the sorting will not be done successfully.

For the last one, although it is not a limit basically, I want to mention it here. It is that our program could not distinguish the black and white objects successfully. We set the received “min” value as 960 to distinguish these two objects, and the best result still has one error.

Experience and Recommendations

In the sorting system design project, the acceleration is not high enough. To improve the stepper acceleration, we can have more code editing and testing. To increase the sorting speed, we can also apply different speeds to the DC motor. However, the speed may affect the sorting accuracy due to the sensor's reading speed.
Besides increasing the speed of the sorting system by the software, we can replace the stepper motor with a more winding phase, so that it can support a stronger torque, enhance the acceleration profile for the stepper motor and increase the operating speed.

The other suggestion that prevents the sorting accuracy affected by the DC motor acceleration is to replace the conveyor belt with a belt with more friction. When the DC motor is at a very high speed, the object might be thrown away at the end of the belt due to inertia. If this specific inertia problem is fixed, it will improve the performance of the system.

Lastly, to make the sorting system more accurate, we can add a physical low pass filter circuit or add some digital signal processing program. This change may eliminate the noise from the ADC. Although we have such a setup in the circuit, the ADC readings of the white plastic and black plastic may also have the same range.

References

[1] Z. Kunwu, Class Lecture, Topic: “MECH458_Final Project_2022Spring”, Faculty of Engineering, University of Victoria, Victoria, BC., Feb., 2022.

Appendix. A Main Code (C)

```
#include <avr/interrupt.h>
#include <stdlib.h>
#include <avr/io.h>
#include <string.h>
#include "lcd.h"
#include "LinkedQueue.h"

// define the global variables that can be used in every function  ==========

volatile unsigned int ADC_result; //char

// 0b0000/INA/INB/ENA/ENB

#define MOTOR_BRKVCC 0b00001111

#define MOTOR_CW  0b00001011

#define MOTOR_CCW 0b00000111

#define MOTOR_BRKG 0b00000011

#define STEP1 0b00110101 //0b00110000
#define STEP2 0b00110110
#define STEP3 0b00101110
#define STEP4 0b00101101
// #define STEP1 0b00110000; //0b00110000
// #define STEP2 0b00000110;
// #define STEP3 0b00101000;
// #define STEP4 0b00000101;


int stepnumber;
int m;

// Global variable for checking the state of the motor and the kill ISR

volatile unsigned int state_Flag=1 ;

volatile unsigned int interruptflag = 0;
volatile int max;
volatile int min=1200;
```

```
volatile unsigned int ADC_result_flag;
volatile unsigned int PRFlag;
volatile int count;
volatile int tim3_count;
volatile int curPos=0;
volatile int futPos;
volatile int del_Q_flag;
volatile int ramp_flag;
volatile int rotate_check=0;
//Global variables to display the sorted objects and unsorted objects
volatile int B_Sorted;
volatile int B_Total;
volatile int W_Sorted;
volatile int W_Total;
volatile int A_Sorted;
volatile int A_Total;
volatile int S_Sorted;
volatile int S_Total;
volatile int store2;
volatile int store3;
volatile int array[50];
volatile int
array100[]={26,25,24,22,19,16,13,10,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,
7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7
,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,7,10,13,16,19,22,24,25,26};
volatile int count2 = 0;
volatile int count3 = 0;
volatile int count4 = 0;
//Global variables for the linked list
 link *head;                           /* The ptr to the head of the queue */
 link *tail;                           /* The ptr to the tail of the queue */
 link *newLink;            /* A ptr to a link aggregate data type (struct) */
 link *rtnLink;            /* same as the above */
 element eTest;                        /* A variable to hold the aggregate data type known as element
*/


void Hall();
void Rota_CW();
void Rota_CCW();
void mTimer(int count);
void pwmDEF();


void main()
{

CLKPR=0x80;
```

```
CLKPR=0x01; //Required to set CPU Clock to 8MHz

TCCR1B|= _BV(CS11);

cli();  // disable all of the interrupt ==========================

// config the external interrupt ======================================

        //Enable INT2, INT3, INT4, INT5
        EIMSK |= (_BV(INT2) | _BV(INT3) | _BV(INT4) | _BV(INT5));

        //Rising edge interrupt for INT2 (OR); Falling edge interrupt for INT3 (EX), INT4
(Pause-Resume Button), INT5 (Ramp-Down Button)
        EICRA |= (_BV(ISC21) | _BV(ISC20) | _BV(ISC31));
        EICRB |= (_BV(ISC41) | _BV(ISC40)| _BV(ISC51)) ;

// config ADC ==========================================================
        //enable ADC; enabale ADC interrupt; set a pre-scaler of 32 for ADC (ADC got a
frequency of 250kHz)
        ADCSRA |= (_BV(ADEN) | _BV(ADIE) );

        //The ADC is right-adjusted; ADC1 is configured
        ADMUX |= (_BV(REFS0) | _BV(MUX0));


// sets the Global Enable for all interrupts ==========================

sei();
// initialize the ADC, start one conversion at the beginning ==========

        InitLCD(LS_BLINK|LS_ULINE);

        DDRC = 0xFF; // set the PORTC as output to display the ADC result
==================
        DDRE = 0x00; //initialize the port E as the input
        DDRD = 0x00;
        DDRL = 0xFF;
        DDRB = 0xFF; //initialize the port B
        DDRA = 0b00111111; //0xFF
        initLink(&rtnLink);
        rtnLink = NULL;
        newLink = NULL;

        pwmDEF();
        Hall();
        curPos=0;

   PORTB = MOTOR_CCW;

                while(1){
```

```
if ((PIND&0x04)==0 && min < 1200)
{
        ADC_result_flag = 1;
                //array[count2]=min;

}

   if (ADC_result_flag){
          //min=array[0];
              //Set on the flag to display the result on LCD
              if (min<= 300){
                        array[count2]=1;//AL
                        count2++;

                    A_Total = A_Total + 1;
                        A_Sorted=A_Total-1;
              }
              else if ( (400 <= min) && (min <= 800))
              {
                    array[count2]=3;//S
                        count2++;
                    S_Total = S_Total + 1;
                         S_Sorted=S_Total-1;

              }
              else if( (870 <= min) && (min <= 963))
              {
                    array[count2]=2;// W
                        count2++;
                    W_Total = W_Total + 1;
                         W_Sorted=W_Total-1;
              }
              else if( (964 < min) && (min <= 1000))
              {
                    array[count2]=0;//B
                        count2++;
                    B_Total = B_Total + 1;
                         B_Sorted=B_Total-1;
              }
                  // LCDClear();
                   //LCDWriteIntXY(0,0,array[0],4);
                   //LCDWriteIntXY(6,0,array[1],4);
                   //LCDWriteIntXY(0,1,array[2],4);
                  // LCDWriteIntXY(6,1,array[3],4);
              LCDWriteIntXY(11,0,min,4);
                    LCDWriteIntXY(11,1,count4,4);
                    //mTimer(1000);
            ADC_result_flag = 0;
            min = 1200;
```

```
                }


                        if (del_Q_flag == 1){
                                del_Q_flag = 0;
                                if(count2==8){
                                        count2=0;
                                }
                                count3++;
                                if(count3==8){
                                        count3=0;
                                }


                        }
                                if (ramp_flag){

                                         TCCR3B = 0x00;
                                         TIMSK3 = 0x00;
                                        cli();
                                        while(1){
                                                ramp_flag = 0;
                                         }
                                }

                        //LCDWriteIntXY(6,0,min,4);

                }


}   // end main


/*************************************************************************
INT2 (OR sensor)
*************************************************************************/
ISR(INT2_vect){
        //PORTL = 0x0F;
        PORTL = 0b11000000;

        if ((PIND&0x04)==0x04)
        {
                ADCSRA |= _BV(ADSC);
        }
}
/*************************************************************************
Bad ISR
*************************************************************************/
ISR(BADISR_vect){
```

```
        LCDWriteString("BADCALLED");
        while(1);
}
/*
INT3 EX sensor
*/

ISR(INT3_vect)
{

                del_Q_flag = 1;
                PORTL = 0b00110000;
      PORTB = MOTOR_BRKVCC;
      mTimer(20);
                futPos = array[count3];


                rotate_check = futPos - curPos;
                curPos = futPos;
                 if ((rotate_check == 2) || (rotate_check == -2))
                 {
                          Rota_CW(100);


                 }
                 else if ((rotate_check == 1) || (rotate_check == -3))
                 {
                          Rota_CW(50);

                 }
                 else if ((rotate_check == -1) || (rotate_check == 3))
                 {
                          Rota_CCW(50);

                 }
                LCDClear();
                LCDWriteIntXY(0,0,array[0],4);
                LCDWriteIntXY(6,0,array[1],4);
                LCDWriteIntXY(0,1,array[2],4);
                LCDWriteIntXY(6,1,array[3],4);


                 /*eTest = firstValue(&head);
                 futPos = eTest.itemCode;
                 rotate_check = futPos - curPos;

      if ((rotate_check == 2) || (rotate_check == -2))
      {
         Rota_CW(100);
                          rotate_check = 0;
```

```
        }
        else if ((rotate_check == 1) || (rotate_check == -3))
        {
           Rota_CW(50);
                        rotate_check = 0;
        }
        else if ((rotate_check == -1) || (rotate_check == 3))
        {
           Rota_CCW(50);
                        rotate_check = 0;
        }

                        dequeue(&head, &tail, &rtnLink);
         curPos = futPos;*/
                    if(array[count3]==0){
                        B_Sorted++;
                    }
                    if(array[count3]==2){
                        W_Sorted++;
                    }
                    if(array[count3]==3){
                        S_Sorted++;
                    }
                    if(array[count3]==1){
                        A_Sorted++;
                    }
                   mTimer(150);
                   PORTB = MOTOR_CCW;

}
// sensor switch:  Active HIGH starts AD converstion =======

// the interrupt will be trigured if the ADC is done ========================

ISR(ADC_vect)  {
   if ((PIND&0x04)==0x04)
   {
      ADCSRA |= _BV(ADSC);

   }
   if(ADC <= min)
   {
      min = ADC;
   }
}
//pause and resume button
ISR(INT4_vect)

          {
                    if((PINE&0x10)==0x10){// push
```

```
                        mTimer(10);
                        if(state_Flag ){

                                state_Flag = 0;
                                mTimer(20);
                                PORTB = MOTOR_BRKVCC;
                                mTimer(10);
                                 LCDClear();

                                 LCDWriteStringXY(0, 0, "Bl:");
                                 LCDWriteIntXY(3,0,B_Sorted, 2);
                                 LCDWriteIntXY(5,0,B_Total-B_Sorted,2);
                                 LCDWriteString(" ");

                                 LCDWriteString("Al:");
                                 LCDWriteIntXY(11,0,A_Sorted, 2);
                                 LCDWriteIntXY(13,0,A_Total-A_Sorted,2);

                                 LCDWriteStringXY(0, 1, "Wh:");
                                 LCDWriteIntXY(3,1,W_Sorted, 2);
                                 LCDWriteIntXY(5,1,W_Total-W_Sorted,2);

                                 LCDWriteString(" ");

                                 LCDWriteString("St:");
                                 LCDWriteIntXY(11,1,S_Sorted, 2);
                                 LCDWriteIntXY(13,1,S_Total-S_Sorted,2);


                        }
                        else {

                                state_Flag = 1;
                                mTimer(20);
                                PORTB = MOTOR_CCW;
                                mTimer(10);

                        }
                        while((PINE&0x10)==0x10){} // push
                        mTimer(20);
                }
}
//Ramp-down ISR
ISR(INT5_vect){
   //Initialize a second timer to count 5 seconds
   //Set the pre-scaler of timer 3 to 256
   TCCR3B |= _BV(CS32)|_BV(CS30);

   //Set the timer 3 mode to CTC for comparing OCRA
```

```
   TCCR3B |= _BV(WGM32);
         //enable timer interrupt
         TIMSK3|= _BV(OCIE3A);
   //Set the output time to 5sec, count for every 1 second.
   OCR3A = 0x989E;
   //Set the counter to 0 and enable the interrupt
   TCNT3 = 0x0000;
}

ISR(TIMER3_COMPA_vect){


      PORTB = MOTOR_BRKVCC;

      LCDClear();

      LCDWriteStringXY(0, 0, "Bl:");
      LCDWriteInt(B_Sorted, 3);
      LCDWriteString("   ");

      LCDWriteString("Al:");
      LCDWriteInt(A_Sorted, 3);

      LCDWriteStringXY(0, 1, "Wh:");
      LCDWriteInt(W_Sorted, 3);
      LCDWriteString("   ");

      LCDWriteString("St:");
      LCDWriteInt(S_Sorted, 3);

      ramp_flag = 1;
      PORTB = MOTOR_BRKVCC;

}




void Rota_CW(int turndegree){
         int delay = 0;
         for (int i = 0; i < turndegree; i++){
                  stepnumber++;
                  if(stepnumber == 5){
                           stepnumber=1;
                  }
                  switch (stepnumber)
                  {
                           case 1:
                           PORTA = STEP1;
                           mTimer(array100[delay]);
```

```
                        delay++;
                        break;
                        case 2:
                        PORTA = STEP2;
                        mTimer(array100[delay]);
                        delay++;
                        break;
                        case 3:
                        PORTA = STEP3;
                        mTimer(array100[delay]);
                        delay++;
                        break;
                        case 4:
                        PORTA = STEP4;
                        mTimer(array100[delay]);
                        delay++;
                        break;
                }


        }
}

/*
To do: counterclockwise rotate when stepnumber less than equal to 0, it must start from 4,
otherwise, it start from 4 to 1.
*/
void Rota_CCW(int turndegree){
        int delay = 0;
        for (int i = 0; i <turndegree; i++){
                stepnumber--;
                if(stepnumber <= 0){
                        stepnumber = 4;
                }

                switch (stepnumber)
                {
                        case 1:
                        PORTA = STEP1;
                        mTimer(array100[delay]);
                        delay++;
                        break;
                        case 2:
                        PORTA = STEP2;
                        mTimer(array100[delay]);
                        delay++;
                        break;
                        case 3:
                        PORTA = STEP3;
                        mTimer(array100[delay]);
```

```
                        delay++;
                        break;
                        case 4:
                        PORTA = STEP4;
                        mTimer(array100[delay]);
                        delay++;
                        break;
                }
                }
        }



/*****************************************************************************************/
/***************************** SUBROUTINES ***********************************************/
/*****************************************************************************************/
/*****************************************************************************
* DESC: initializes the linked queue to 'NULL' status
* INPUT: the head and tail pointers by reference
*/

void setup(link **h,link **t){
    *h = NULL;          /* Point the head to NOTHING (NULL) */
    *t = NULL;          /* Point the tail to NOTHING (NULL) */
    return;
    }/*setup*/
    /*****************************************************************************************
    * DESC: This initializes a link and returns the pointer to the new link or NULL if error
    * INPUT: the head and tail pointers by reference
    */
 void initLink(link **nl){
//link *l;
 *nl = malloc(sizeof(link));
 (*nl)->next = NULL;
 return;
}/*initLink*/

/*****************************************************************************************
*  DESC: Accepts as input a new link by reference, and assigns the head and tail
*  of the queue accordingly
*  INPUT: the head and tail pointers, and a pointer to the new link that was created
will put an item at the tail of the queue */
void enqueue(link **h, link **t, link **nL){

if (*t != NULL){
/* Not an empty queue */
 (*t)->next = *nL;
 *t = *nL; //(*t)->next;
}/*if*/
else{
```

```
    /* It's an empty Queue */
    //(*h)->next = *nL;
    //should be this
 *h = *nL;
 *t = *nL;
 }/* else */
return;
}
/*enqueue*/

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

//*****************************************************************************************
* DESC: Peeks at the first element in the list
* INPUT: The head pointer
* RETURNS: The element contained within the queue
*/
/* This simply allows you to peek at the head element of the queue and returns a NULL
pointer if empty */
element firstValue(link **h){
    return((*h)->e);
    }/*firstValue*/

/*****************************************************************************************
* DESC: deallocates (frees) all the memory consumed by the Queue
* INPUT: the pointers to the head and the tail
*/
/* This clears the queue */
void clearQueue(link **h, link **t){

    link *temp;

    while (*h != NULL){
        temp = *h;
        *h=(*h)->next;
        free(temp);
        }/*while*/

        /* Last but not least set the tail to NULL */
        *t = NULL;

        return;
        }/*clearQueue*/

/*****************************************************************************************
* DESC: Checks to see whether the queue is empty or not
* INPUT: The head pointer
* RETURNS: 1:if the queue is empty, and 0:if the queue is NOT empty
*/
/* Check to see if the queue is empty */
```

```
char isEmpty(link **h){
   /* ENTER YOUR CODE HERE */
   return(*h == NULL);
   }/*isEmpty*/

/*****************************************************************************
* DESC: Obtains the number of links in the queue
* INPUT: The head and tail pointer
* RETURNS: An integer with the number of links in the queue
*/
/* returns the size of the queue*/
int size(link **h, link **t){

   link  *temp;                  /* will store the link while traversing the queue */
   int   numElements;

   numElements = 0;

   temp = *h;                    /* point to the first item in the list */

   while(temp != NULL){
      numElements++;
      temp = temp->next;
      }/*while*/

      return(numElements);
   }

void mTimer(int count){

         int i;

         i =0; /* initializes loop counter */

/* Set the Wavefom Generation mode bit description to Clear Timer on Compare Math mode
(CTC) only */

         TCCR1B |=_BV(WGM12);

/* set WGM bíts to 0100, see page 145 */

/*Note WGN is spread over two register. */

         OCR1A= 0x03E8; /* Set Output Compare Register for 1000 cycles =1ms */

         TCNT1 = 0x0000; /* Sets initial value of Timer Counter to 0x0000 */

//TIMSK1=TIMSK1 |0b00000010; /* Enable the output compare interrupt enable */

         TIFR1 |= _BV(OCF1A); /*clear the timer interrupt flag and begin new timing */
```

```
/* If the following statement is confusing, please ask for clarification! */

/* Poll the timer to determine when the timer has reached 0x03E8 */

        while(i<count){

                if((TIFR1 &0x02)==0x02){

                TIFR1 |=_BV(OCF1A); /* clear interrupt flag by writing a ONE to the bit */

                i++;/*increment loop number*/

                }

        }

return;

}

void pwmDEF(){
        //Set the timer to fast PWM mode; pg 126
        TCCR0A |= _BV(WGM00) | _BV(WGM01);
        //Clear on a compare match; Set output compare A when the time reaches TOP; pg
126
        TCCR0A |= _BV(COM0A1);

        //Set the frequency for timer 0 for PWM with a pre-scaler of 1024
        TCCR0B |= _BV(CS01) | _BV(CS00);
        //Set the duty cycle for PWM; 50%, x/255
        OCR0A = 0x55; //SPEED
}
void Hall(){
   while ((PINA&0x80) == 0x80)
   {
      stepnumber++;
      if(stepnumber == 5){
         stepnumber=1;
      }
      switch (stepnumber)
      {
         case 1:
         PORTA = STEP1;
         mTimer(20);
         break;
         case 2:
         PORTA = STEP2;
         mTimer(20);
         break;
```

```
            case 3:
            PORTA = STEP3;
            mTimer(20);
            break;
            case 4:
            PORTA = STEP4;
            mTimer(20);
            break;
        }
    }
    curPos = 0;
}
```

Appendix. B Circuit Connection