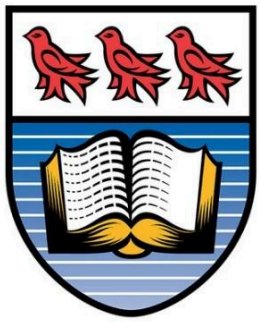
University of Victoria
Faculty of Engineering
MECH 410
Fall 2023

MECH 458



University of Victoria

Project Report

Created By:

Colm Molder

V00937879

Blaine Tubungbanua

V00918128

Table of Contents

List of Figures

List of Tables

1 Project Introduction

The MECH 458 project is centred around the development of a mechatronic sorting system using C programming on an ATmega2560. The ATmega2560 is an 8 bit micro controller with 256KB of Flash program memory [1]. The microcontroller features 12 channel 16-bit resolution PWM channels and A/D-converter as well as a 16 Mhz crystal system clock. All these features are critical to the project.

The microcontroller will be used to control a mechatronic conveyor belt sorting system seen below in Figure 1.

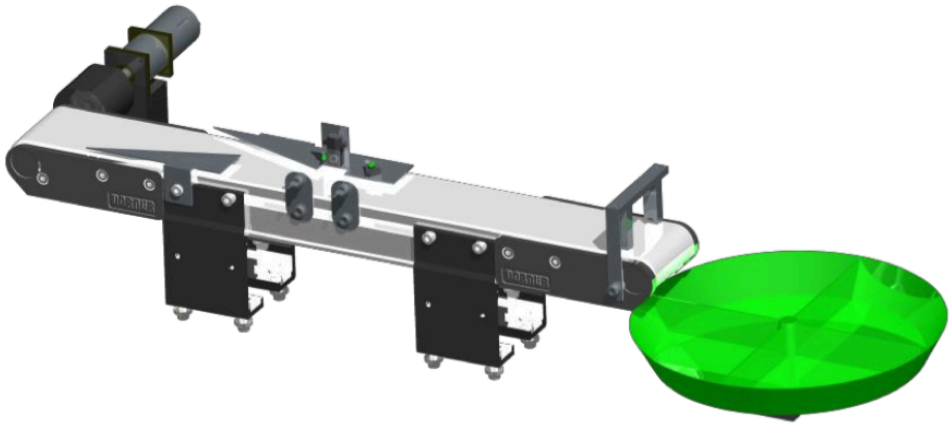


Figure 1: Mechatronic Sorting System [1]

The system features a ferromagnetic sensor (which is not required for this project), a reflectometer sensor, a laser optical sensor at each of these, and an end of travel optical sensor. The conveyor belt is controlled by a DC motor and the end bucket is rotated by a stepper motor. The stepper motor position is homed through a sensor on the bucket.

The assignment is to sort 40 objects into separate bins. These objects are either white plastic, black plastic, steel, or aluminium. The performance of the system is determined by the number of correct sorts (Nc), number of errors (Ni) and time to sort all 40 (T).

$$SPI = \frac{N_c - N_i}{T}$$

Beyond sorting, the system must also make use of one push button to pause and resume the system and another push button to ramp down (sort the remaining items on the belt then stop).

Based off the project background discussion above and discussions with the Lab Technician Patrick, several objectives were identified:

1. System must sort 48 objects in under 60 seconds
2. System must make no more than 1 error
3. System must have pause functionality and display the number of each sorted material/colour and the number of objects still on the belt
4. System must have ramp down functionality and display the number of each sorted material/colour
5. System must aim to maximise SPI score
6. System wiring must be clean and easy to track
7. System coding must be clean and well laid out and commented

2 System Technical Information

2.1 High Level Function Diagram

A High level function diagram is shown in Figure 2. A detailed description of each sub-process block is described in section 2.3.

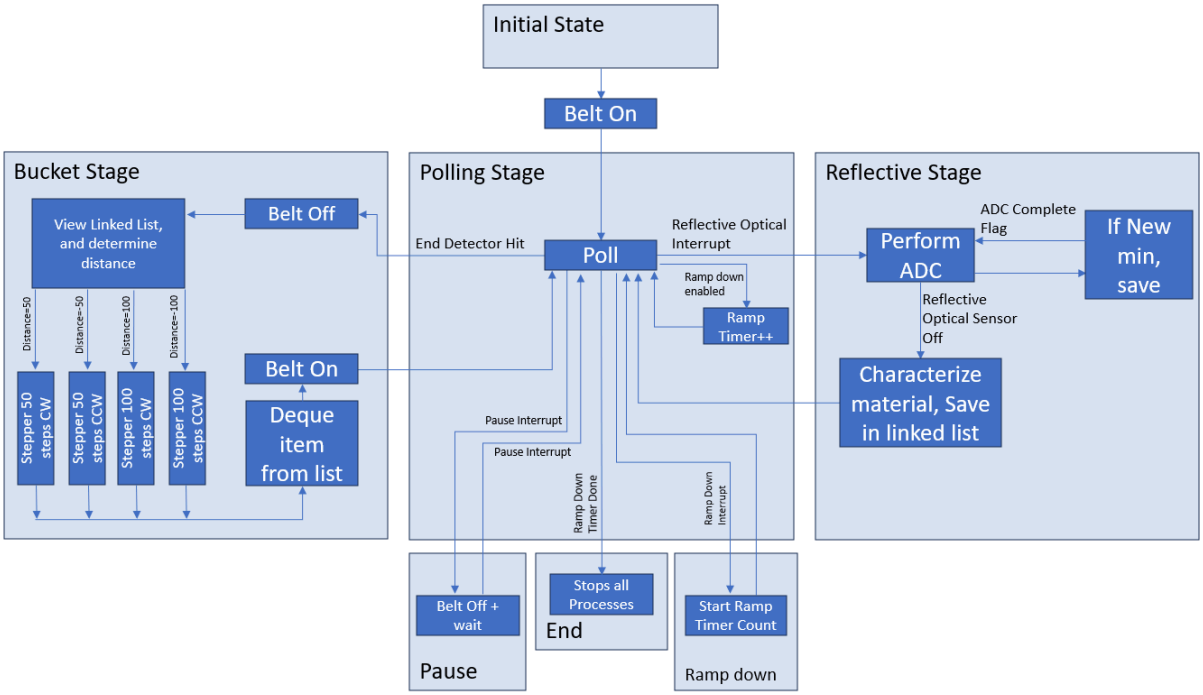


Figure 2: Block Diagram

2.2 Circuit Diagram

The circuit diagram for the system is shown in Figure 3. At the centre is the ATMega 2560 Microcontroller, connected to 4 sensors, 2 actuators, 2 buttons, along with LED and LCD displays. Not included in the circuit diagrams are 47KΩ resistors wired in series with the exit sensor and the reflectance detector, acting as low-pass filters, to eliminate false triggers caused by electrical noise generated by the brushed DC motor.

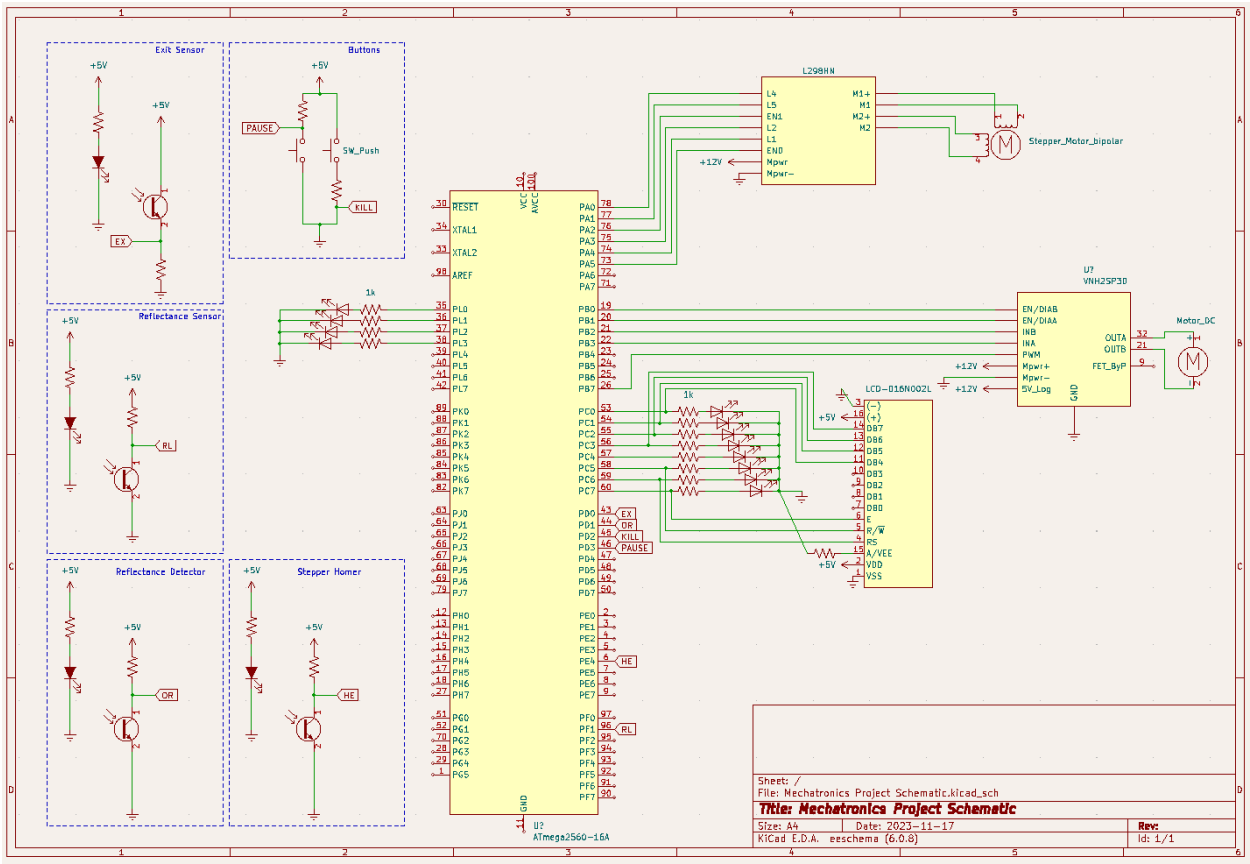


Figure 3: Circuit Diagram

With the exception of the reflectance sensor and hall effect sensor, each sensor is either a pull-up or pull-down signal triggering an interrupt service routine (ISR). These correspond to the ISR watching for a rising edge or falling edge. The relevant sensors are tabulated in Table 1.

- **Reflectance Sensor:** The reflectance sensor is responsible for distinguishing the four parts: Aluminium, Steel, White, and Black. It works by shining a light on the part, and measuring how much light is reflected back. This yields an analog value that must be translated for the MCU with an ADC.
- **Reflectance Detector:** The reflectance detector is responsible for signalling when there is a part in front of the reflectance sensor, signalling for the start and continuation of ADC conversions. Without the reflectance detector, the MCU would constantly conduct

ADC conversions, slowing the performance of the processor. This sensor yields a logic high signal that is pulled down when the beam is broken.

- ➢ **Exit Sensor:** The exit sensor signals when a part reaches the end of the belt, and is about to be ejected into the bucket. This sensor also functions using a beam-break sensor.
- ➢ **Stepper Homer:** The stepper homer defines the home position of the stepper motor. It functions using a hall effect sensor, detecting the magnetic field induced by a magnet placed at the home position. This yields an analog signal that trips into logic high as the magnet passes over the sensor, terminating a while() loop during the stepper homing process.
- ➢ **2x Button:** Two buttons are included to allow the operator to pause, as well as ramp down the system. One button is a pull-up signal, and the other is pulldown.

Table 1: Sensor Types

Sensor	Purpose	Rising/Falling Edge	Sensor Type
Reflectance Sensor	Identify Part Type	N/A	Photoresistor
Reflectance Detector	Signal when a part is present in the reflectance detector zone	Falling Edge	Beam-Break
Exit Sensor	Signal when a part reaches the end of the belt	Falling Edge	Beam-Break
Stepper Homer	Define the stepper motor home	N/A	Hall-Effect
2x Button	Pause / Rampdown	Falling / Rising	Momentary Switch

2.3 System Algorithm

The system is divided into 6 states, yielding a finite state machine. The polling state implements a switch statement that responds to changes in STATE, which are enacted by ISRs, or logic in other states. Within the switch statement a goto statement is used to navigate to the different

states. The full code, including register settings is available in Appendix B. A breakdown of each stage is shown in Table 2.

Table 2: Six Finite State Machine States

State	Purpose
Polling Stage	Default state.
Reflective Stage	Logic for reading ADC values from Reflective sensor, and categorising incoming items.
Bucket Stage	Logic for mechanically sorting items into relevant bins at the end of the belt.
Pause Stage	Standby State, displays items that have been sorted.
Ramp Stage	Transient State initialising conditions to begin system rampdown.
End Stage	End State. System enters a safe state and stops all processes.

2.3.1 Polling Stage:

The polling stage is the default state of the system. It continually watches for system changes, and uses a goto statement within each case, to navigate to each `STATE.` Additionally, there is a latent timer embedded in the polling state. If the ramp down has been initialised, it begins running for an allotted time before navigating to the END state.

```
POLLING STAGE:
     if(ramp down has been initialised)){
          Decrement countdown timer, based on clock
          if(countdown complete){
               Set state to END
          }
     }
     switch(STATE){
          if(state 1) goto POLLING_STAGE
          if(state 2) goto PAUSE_STAGE
```

```
        if(state 3) goto REFLECTIVE_STAGE
        if(state 4) goto BUCKET_STAGE
        if(state 5) goto RAMP_STAGE
        if(state 6) goto END_STAGE
    }
```

2.3.2 Reflective Stage:

The reflective stage, triggered by the reflectance detector, identifies, categorises, and stores the incoming items in a FIFO linked list. Dealing with the reflectometer requires management of the ADC. The ADC runs the conversion in the background. Once the conversion is complete, an ISR is triggered, the ADC_result_flag is set to true, and the result is stored in the ADC register. The reflectance value reaches its minimum at the centre of the part. Because of this, only the minimum reflectance value is stored. Different item categories have different reflectance values ranging from 1 to 1024. Calibration is discussed in section 4. If an item has a reflectance within a certain range, it is organised into that category.

```
REFLECTIVE STAGE:
     if(an ADC reading has been completed){
          if(The value is also a new minimum){
               Overwrite the last stored value
          }
          Once complete, make a new ADC conversion request
     }

     Once the part is no longer being scanned, examine the minimum
     value, and categorise the part

    if(category_min < itemVal < category_max){
        Assign relevant ID
        Increment # of counted parts
    }
```

2.3.3 Bucket Stage:

The bucket stage is triggered by the end detector. It acknowledges the approaching parts, and rotates the bucket to allow them to fall into the correct bin. The stepper motor has 200 steps for a full rotation, corresponding to relative destinations at +50 steps, -50 steps, and +100 steps, with +/- 100 steps arbitrarily chosen. The system halts while the stepper completes a whole spin cycle, simplifying the code, but limiting the ability for synchronous sorting and ADC conversions.

```
BUCKET_STAGE:
     Pause the belt
     View the item in the linkedList
     Determine the destination position (will be A,S,W,B)
     Compute the distance from current position to destination
     ————— position via CW and CCW rotation
     Compare the distance to destination via CW and CCW rotation
      Select the shortest distance (will be +50, +100, +150, or -50,
     ————— -100, -150) decide which direction to turn
      rotateStepper(By +50, or -50, or +/-100 steps)

     delay(bucketFall) //give enough time for the previous part to
     ————land before rotating again
      Resume the belt
      Remove the item from the queue
      -- itemCategoryOnBelt_count
     ++sortedItemCategory_count
```

2.3.4 Pause Stage

The pause stage is entered when the pause button is pressed, calling an ISR that sets the STATE to pause. The pause state pauses the belt, and displays all the items on the belt, as well as all the items that have been sorted.

```
PAUSE_STAGE:
    Listen for pause-button-activation via ISR.
    debounce()
```

```
    if(current state is not yet paused){
        Pause the belt
        Clear the LCD
        Display all items on belt
        Display all items that have been sorted
    }
    else if(current state is already paused){
        Clear the LCD
        Resume the belt
    }
```

2.3.5 Ramp Stage

The ramp stage is a shutdown sequence that, when pressed, sorts all the remaining items on the belt before shutting down. The ramp stage is entered when the ramp down button is pressed. The ramp stage implementation initialises a countdown that runs in the POLLING state, by setting a ramp down boolean. The countdown is reset every time a new part is scanned, leaving enough time for the belt to be cleared before shutting down. Because the countdown only runs in the POLLING state, when the bucket is moving, the countdown is paused. This is a system feature that helps to ensure that all parts are sorted from the belt before shutting down. Once the countdown is complete, the system switches to the END state (section 2.3.6). See section 2.3.1 for the countdown timer pseudocode.

```
RAMP_STAGE:
    debounce()
    if(ramp down has not yet been initialised){
        Set ramp down boolean to true
    }
    else if(ramp down has already been initialised){
        Set ramp down boolean to false
    }

ISR(INT1_vect){ //reflective sensor interrupt
    Reset rampdown timer
}
```

2.3.6 End Stage

The end stage is the final state of the system, entered when the ramp down sequence has been initialised, and completed, with all the items sorted from the belt. This stage stops the belt, displays all the sorted items on the LCD, and exits main().

```
END:
    StopTheBelt()
    LCDClear()
    Display  all sorted parts
    return 0
```

3 System Performance Specifications

The final results of the system was the successful sorting of 48 parts in 39 seconds with 0 errors. Parts were loaded eight at a time. This converts to a system performance index (SPI) of 1.23. Where SPI is given by:

$$SPI = \frac{N_c - N_i}{T} = \frac{48-0}{39} = 1.23$$

The main errors noted during the final demonstration were caused by a calibration nuance between the end-detector position, and the speed of the belt. On occasion, a part would prematurely fall off the end of the belt before the stepper was able to bring the correct bucket into position. This would result in an offset of readings affecting all remaining parts. Other issues occurred when parts were too spaced out and parts would trip the end detector and reflective sensor at the same time. As will be discussed in section 5 - Limitations and System Tradeoffs, the code is not capable of handling this scenario.

3.1 Recommended Operating Parameters

The calibration procedure to determine operating parameters will be discussed in section 4 - Testing and Calibration Procedures. From these calibration tests, the parameters in Table 3 are

recommended to provide constant and reliable sorting operations with an SPI of 1.23. A definition of each item in Table 3 is listed below.

- Belt speed: The PWM duty cycle supplied to the belt DC motor.
- Counts before ramp down end: The time allocated to complete sorting before shutting down
- Initial Step Delay: The step delay applied to the first step of the stepper acceleration curve
- Steady State Step Delay: The step delay applied to the maximum stepper speed
- Number of Acceleration Steps: The number of steps allocated to the acceleration from initial to max stepper speed
- Number of Deceleration: The number of steps allocated to the deceleration
- Delay From Drop to Stepper: The delay to allow an item to fall before the next sequence
- Stepper Offset Steps: Alignment of the stepper with the belt.

Table 3: Recommended Operating Parameters

Misc.	Belt Speed	0xA0 (63% duty)
	Counts Before Rampdown End	100
Stepper	Initial Step Delay [ms]	20
	Steady State Step Delay [ms]	6
	Number of Acceleration [Steps]	11
	Number of Deceleration [Steps]	6
	Delay From Drop to Stepper [ms]	8
	Stepper Offset steps	7
ADC Ranges	Aluminum Min [ADC Value]	1
	Aluminum Max [ADC Value]	216
	Steel Min [ADC Value]	217
	Steel Max [ADC Value]	735
	White Min [ADC Value]	736
	White Max [ADC Value]	919
	Back Min [ADC Value]	920
	Black Max [ADC Value]	1024

3.2 Maximum Operating Parameters

Through further testing, faster stepper acceleration profiles and belt speeds were possible as seen in Table 4.

Table 4: Maximum Operating Parameters

Misc.	Belt Speed	0xC0 (75% duty)
Stepper	Initial Step Delay [ms]	15
	Steady State Step Delay [ms]	5
	Number of Acceleration [Steps]	8
	Number of Deceleration [Steps]	6

Despite the potential for faster sorting operations, these parameters occasionally lead to losing steps at high loading, and mis-identifying items. As such, they are not recommended for consistent and reliable sorting.

4 Testing and Calibration Procedures

As discussed in section 3.1 - Recommended Operating Parameters, several tuning factor variables were created and are listed in Table 3. Physically, the angle of the end detector was another adjustable variable.

To determine the ideal belt speed, several tests were completed. First, it was determined how fast the belt could run while still achieving enough ADC readings to read all parts without skipping, and record accurate reflectivities. Secondly, the speed was varied to ensure parts at the end detector remained on the belt upon stopping. Through these tests, an ideal belt speed was determined to be powered by a 63% duty cycle. During the belt speed tests, the ideal end detector angle was determined as well. This angle was found by creating the largest angle possible, that still ensured parts would stay on the belt. The final angle is seen below.

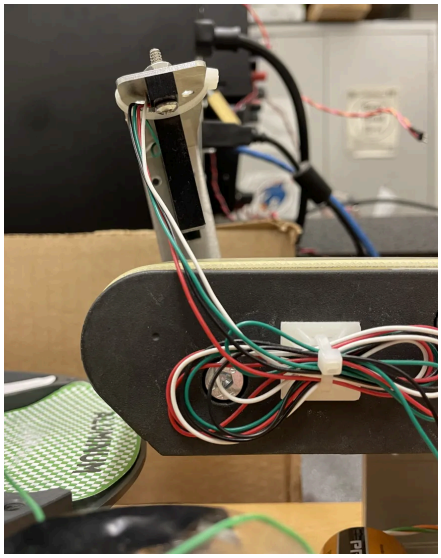


Figure 4: End Detector Angle

To calibrate the stepper motor acceleration profile, tests were completed with the bucket fully loaded, and made to complete repeated 180 degree rotations. Variations were first made to the initial stepper speed to find a value that could move the stepper regardless of loading. The value determined was a delay of 20ms between steps. Next, variations in the steady state speed, and number of acceleration and deceleration steps were made in turn. These values were continually decreased until steps were lost during motion. The results of these tests found an ideal steady state delay of 6ms between steps, 11 acceleration steps, and 6 deceleration steps. The acceleration profile is seen below.

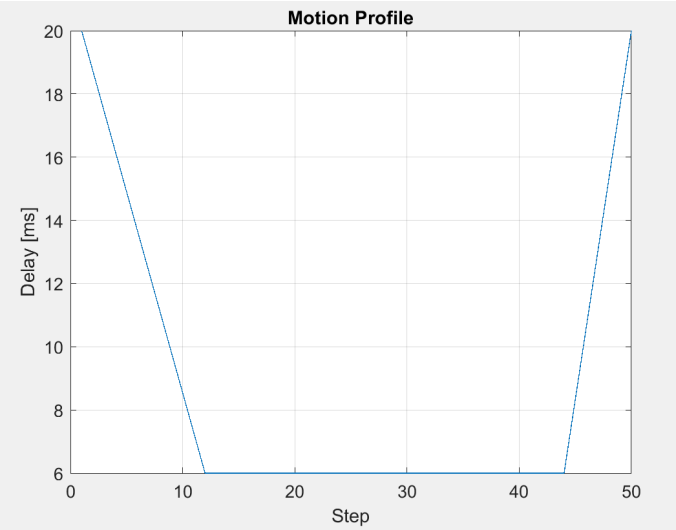


Figure 5: Stepper Acceleration Profile

The delay time from part drop to the start of stepper movement was determined through slow motion video to ensure parts landed in the centre of the bucket before movement began. This delay was determined to be 8ms. Finally, the angle of the bucket was offset by 7 steps to line up the off centre end detector with the centre of the bucket.

The number of counts before ramp down completion was determined by the time required for a part to be read, then sorted. This time will be usable regardless of the number of parts still to be sorted because the ramp down countdown restarts upon part reading and pauses during part sorting. The number of counts was determined to be 100.

The final tuning variables set were the ADC recording ranges to distinguish between white, black, steel, and aluminium parts. These values were determined through a calibration code. The calibration code runs the belt and saves the minimum and maximum ADC recordings during operation. Every part of each type was run through the system to find the absolute maximum and minimum values. Thes ranges were then inflated to meet the next range halfway between them. The final ranges are seen below.

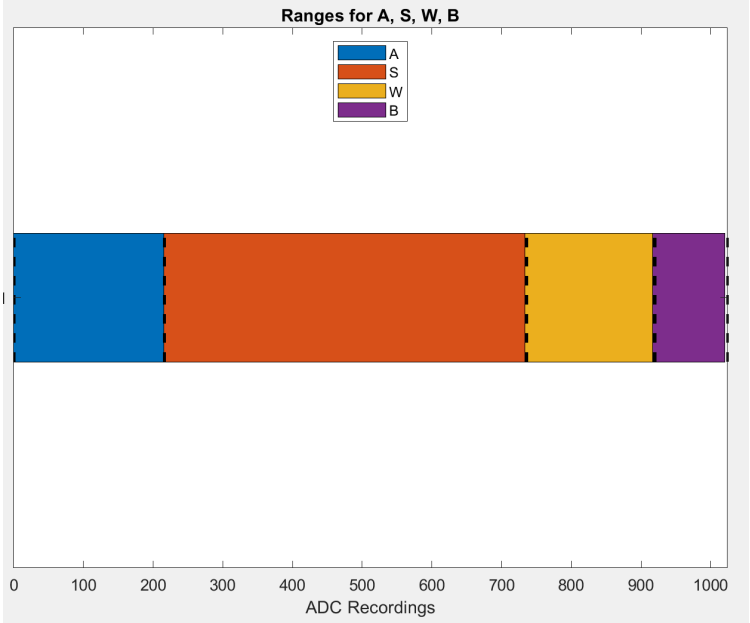


Figure 6: ADC Ranges

Once fully calibrated, rigorous system testing was completed. The LCD was heavily used during this process to display objects saved in the que, and objects sorted. This process was essential to determine if erroneous sorting was the fault of the ADC reading, the stepper positioning, or the handling of the linked list. Various final testing was completed. Initial tests were aimed to ensure the code's logic could handle any possible combination of parts possible and accurately sort. Later tests ensured the code, stepper motor, and reflective sensor could handle a larger volume of parts.

5 Limitations and System Tradeoffs

The main limitation of the design is the inability to fully load the belt, which reduces sorting speed. When the belt is fully loaded, the reflectometer and end detector are triggered while the stepper is simultaneously rotating. This issue was not pursued in favour of other debugging tasks, so the exact mechanism of failure is not yet known, and would require further testing. The most likely cause is the implementation of the bucket stage. When the bucket stage is entered, the logic enters a loop, and all 50 to 100 steps are made within the loop, never allowing the system

to return to the REFLECTIVE_STAGE to read and store ADC values. A potential solution would be to rewrite the system to make a single step after each polling cycle, storing the position and destination in global variables. This method was attempted early in the project, but was abandoned in favour of the current method to accelerate progress to a minimum viable product (MVP).

For the stepper motor acceleration, a trapezoidal curve was used rather than an S-curve, which several other groups implemented. While the S-curve yields a smoother and more rapid acceleration, a trapezoid curve was used based on its reliable usage in industrial CNC applications [2], and to accelerate progress to a MVP.

The accuracy of the system is also dependent on the quality of parts. While the reflectivity of steel, aluminium, black-plastic, and white-plastic each reside in their own distinct ranges, the black and white plastics ranges converge, and can intersect if the parts are reasonably scuffed, which happens over time. This results in missorted black and white parts if the parts are poor quality.

6 Experience and Recommendations

The overall project experience was positive and the group made steady progress through the term culminating in a successful design without the need for large last minute efforts. Approximately four extra hours a week on top of lab time was required to complete the project. Some aspects that lead to the team's success was a good understanding of the technical manual, proficient use of LED’s, multimeters, and the LCD for debugging, and a good working relationship with lab technicians and other groups to provide assistance when stuck. The only recommendation the team has for the lab structure is to provide a more detailed overview of the wiring and setup involved in the conveyor apparatus. This would help students appreciate the process required to create a full mechatronic system as the project focuses mainly on the software side. No further lab structure recommendations are held, other than the continued conveyance of the importance of putting in the required time to finish the project without stress.

Recommended improvements to the apparatus are related to the bucket's handling of parts. During testing, it is difficult to keep track of parts as they fall out of the bucket. Potentially a bucket with deeper walls could alleviate this issue. However, the stepper motor is likely incapable of moving when full of aluminium and steel parts. If this is the case, an alternative method could replace the buckets with open hoops that drop parts straight through into a static basket below. The detriment to this idea is to determine system accuracy, video recording would be required.

7 Conclusion

The project successfully resulted in the development of a mechatronic sorting system, using the ATmega2560 microcontroller to efficiently sort objects based on their material and colour. Demonstrating proficient accuracy, the system accomplished sorting 48 objects within 39 seconds without errors, achieving a System Performance Index (SPI) of 1.23. However, the project was not without its limitations, notably the occasional sorting inaccuracies due to calibration issues between the end-detector position and belt speed, and challenges in handling simultaneous sensor triggering with a fully loaded belt. These areas offer avenues for future work. The project aligns well with the initially stated objectives, achieving technical efficiency while also contributing valuable insights into mechatronics and microcontroller applications. For future projects, it is recommended to focus on enhancing the system's capacity to manage full loads and to explore alternative motor control algorithms, such as the S-curve for stepper speed, to potentially improve system efficiency.

References

[1] UVic Mech 458, “Final Project presentation”, Dr. Yang Shi, Patrick Chang

[2] UVic Mech 460 Lecture Notes, Dr. Keivan Ahmadi

Appendices

A Final System Photo

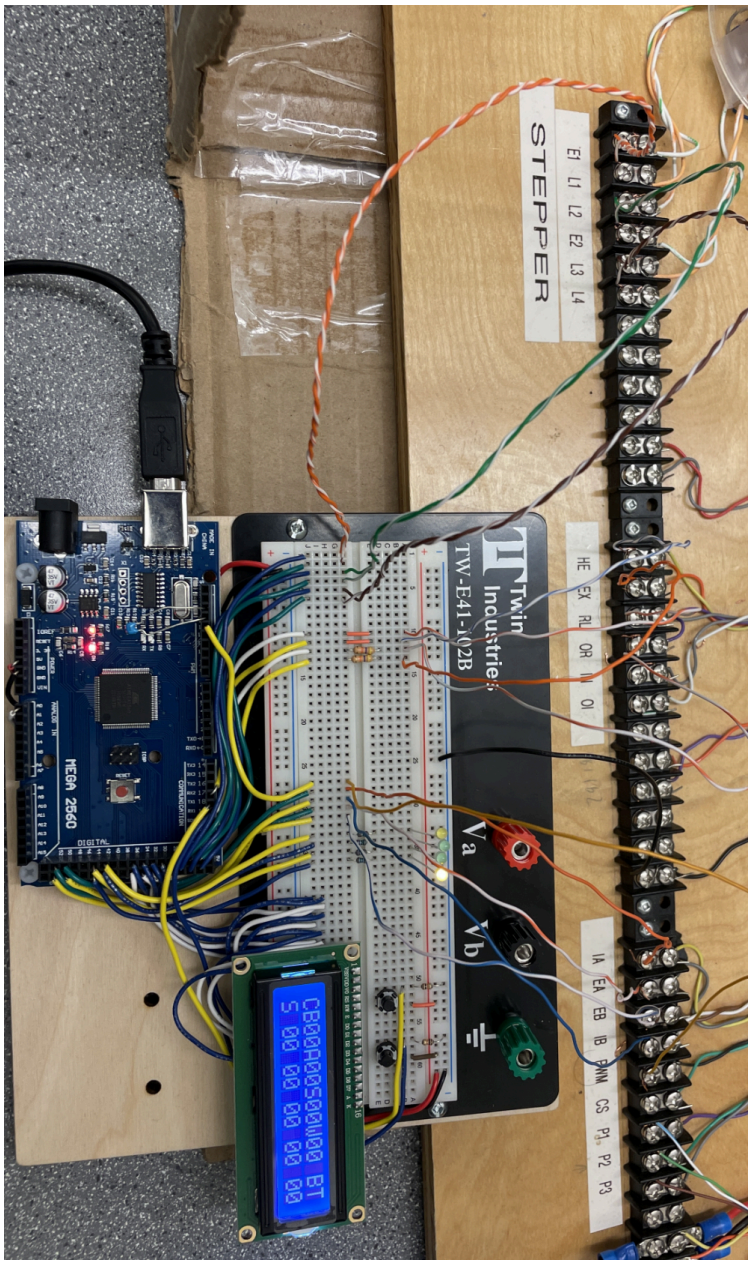


Figure 7: Final System

B System Code

```
//PROJECT CODE TAKE 1
//COLM MOLDER, BLAINE TUBUNGBANUA

//LIBRARIES AND FUNCTIONS AND VARIABLES-------------------------------------------------------------------
#include <avr/interrupt.h>
#include <avr/io.h>
#include <stddef.h>
#include <stdlib.h>
#include "lcd.h"
#include "myutils.h"
#include "LinkedQueue.h"

void mTimer(int count);
int size(link **h, link **t);
void setup(link **h,link **t);
void initLink(link **newLink);
void enqueue(link **h, link **t, link **nL);
void dequeue(link **h, link **deQueuedLink);
element firstValue(link **h);
void clearQueue(link **h, link **t);
char isEmpty(link **h);
int size(link **h, link **t);

void FWD();
void REV();
void BRK();
void KILL();

int initStepper(int n);
int stepperCW(int steps);
int stepperCCW(int steps);
int stepIdx = 1; //arbitrarily 1
int stepperMover(int distance, int stepPos, int Nintey_deg_Profile[], int one_eight_deg_profile[]);

void dispExit(link **head);

//Material limits
#define Alum_MIN 1
#define Alum_MAX 216
#define Steel_MIN 217
#define Steel_MAX 735
#define White_MIN 736
#define White_MAX 919
#define Black_MIN 920
#define Black_MAX 1024

// Global Variables
volatile char STATE;
volatile int Presses=0;
volatile unsigned int ADC_result; //8 bits that store ADC result
volatile unsigned int ADC_result_flag;   //16 bits that flag when ADC conversion complete
int debounce=0;
int pause=0;
int rampdown;
int bTimerInt;
```

```
int bTimer;
int bSubTimer;
int cTimerInt;
int cTimer;
int cSubTimer;



int num_CW=0;//Counted White
int num_SW=0; //Sorted WHite
int num_CB=0;
int num_SB=0;
int num_CS=0;
int num_SS=0;
int num_CA=0;
int num_SA=0;
int num_CT=0;
int num_ST=0;

//START OF MAIN-----------------------------------------------------------------------------------------------

int main(int argc, char *argv[]){
// TUNING FACTORS

                int beltSpeed=0xA0;  //PWM
                int stepperSpeed=6; //Steady-state stepper delay
                int stepperAccel=11; //Number of steps allocated to acceleration
                int stepperDecel=6; //Number of steps allocated to deceleration
                cTimer=100;                     //Number of counts before rampdown closes
                int bucketDelay=8;


// SETTING UP PORTS AND CLOCK AND LCD-------------------------------------------------------------------------------
        CLKPR = 0x80;
        CLKPR = 0x01;           //  sets system clock to 8MHz
        TCCR1B |=_BV(CS11);

        InitLCD(LS_BLINK|LS_ULINE); //initiate the LCD
        LCDClear();
        LCDWriteStringXY(0,0, "Project_V2");

        DDRC = 0xFF;            // just use as a display
        DDRL = 0xFF;            // Green and yellow
        DDRB = 0xFF;            //DC motor
        DDRA = 0xFF;            //Stepper motor
        //Hall effects on port H0
        //interrupts on port D

//SETTING UP TERTIARY TIMER-----------------------------------------------------------------------------
TCCR3B|=_BV(CS31);      //enables prescale. Do we need this to wake up the timer?
TCCR2A |=_BV(WGM32);    //Select CTC
OCR3A = 0xFFFF;                 //Output compare value. Check with documentation for val
TCNT3 = 0x0000;                 //Define counter value to 0x0000
TIMSK3|=_BV(OCIE3A); //Enable the Ouput Compare Match A interrupt


//SETTING UP 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 */
        rtnLink = NULL;
        newLink = NULL;
        setup(&head, &tail);
//SET UP PWM-------------------------------------------------------------------------------------------
        //set PWM mode 3
        TCCR0A |=_BV(WGM01); //setting bit WGM01 to 1 -> sets fast PWM with Top update
        TCCR0A |=_BV(WGM00); //setting bit WGM00 to 1 -> to select mode 3
        TCCR0B |= _BV(CS01); //prescaling the clock
        //Set the compare match output mode to clear, on a compare match
        //And set the output compare A to 1 when the timer reaches TOP
        OCR0A = beltSpeed;
        TCCR0A |= _BV(COM0A1); //Clears on compare match, but sets OC0A at bottom
//SETTING UP Stepper Motors----------------------------------------------------------------------------
        stepIdx = initStepper(90);
        int A = 50;             //Double check that 50 = 45degrees
        int S = 150;
        int W = 100;
        int B = 0;
        int StepPos = 0; //New def: Global position of stepper

//SETTING UP 50 ACCEL
        int total_steps = 50;
        int Accel = stepperAccel;
        int Deccel = stepperDecel;
        int Start_Delay = 20;
        int Min_Delay = stepperSpeed;

        int Steady = total_steps - Accel - Deccel;
        float up_slope = (float)(Start_Delay - Min_Delay) / Accel;
        float down_slope = (float)(Start_Delay - Min_Delay) / Deccel;
        int Nintey_deg_Profile[50];

        int step = 0;

        for (int i = 0; i < Accel; ++i) {
                Nintey_deg_Profile[step] = Start_Delay - (int)(up_slope * i);
                step = step + 1;
        }

        for (int i = 0; i < Steady - 1; ++i) {
                Nintey_deg_Profile[step] = Min_Delay;
                step = step + 1;
        }

        for (int i = 0; i <= Deccel; ++i) {
                Nintey_deg_Profile[step] = Min_Delay + (int)(down_slope * i);
                step = step + 1;
        }


//Setting UP 180 ACCEl
        total_steps = 100;

        Steady = total_steps - Accel - Deccel;
        int One_Eighty_deg_Profile[100];
```

```
        step = 0;

        for (int i = 0; i < Accel; ++i) {
                One_Eighty_deg_Profile[step] = Start_Delay - (int)(up_slope * i);
                step = step + 1;
        }

        for (int i = 0; i < Steady - 1; ++i) {
                One_Eighty_deg_Profile[step] = Min_Delay;
                step = step + 1;
        }

        for (int i = 0; i <= Deccel; ++i) {
                One_Eighty_deg_Profile[step] = Min_Delay + (int)(down_slope * i);
                step = step + 1;
        }


//SETTING UP ADC and interrupts-------------------------------------------------------------------------------
        cli();          // Disables all interrupts
        EICRA |= _BV(ISC21); // --> rising egde| _BV(ISC20); //INT2
        EICRA |= _BV(ISC31); // --> rising egde| _BV(ISC30); //INT3
        EICRA |= _BV(ISC11) | _BV(ISC10); //INT1
        EICRA |= _BV(ISC01); //INT0    for rising edge->  | _BV(ISC00)
        // See page 112 - EIFR External Interrupt Flags
        EIMSK |= 0x0F; //turns on interrupt flags

        ADCSRA |= _BV(ADEN); // ADC Control status register - enable ADC
        ADCSRA |= _BV(ADIE); // enable interrupt of ADC
        ADMUX |= _BV(REFS0); //removed ADLAR
        // Enable all interrupts
        sei();  // Note this sets the Global Enable for all interrupts

//START OF LOGIC-----------------------------------------------------------------------------------------------------------------------
        STATE = 0;
        int min_adc=0xfff; //setting the initial lowest reading (a very large number)

        mTimer(20);
        LCDClear();
        REV();

        goto POLLING_STAGE;

// POLLING STATE-----------------------------------------------------------------------------
        POLLING_STAGE:

                if(rampdown){

                        if(cTimerInt){  //If TIFR has been tripped
                                if(cTimer > 0){
                                        --cTimer;
                                        LCDWriteIntXY(14,1,(cTimer/10),2);
                                }else{
                                        STATE=5;
                                }
                                TIFR3|=_BV(OCR3A);
                                cTimerInt=0;
                        }
                }
```

```
	switch(STATE){
		case (0) :
			PORTL=0b00010000;
			goto POLLING_STAGE;
			break;
		case (1) :
			goto PAUSE_STAGE;
			break;
		case (2) :
			PORTL=0b01000000;
			goto REFLECTIVE_STAGE;
			break;
		case (3) :
			PORTL=0b01000000;
			goto BUCKET_STAGE;
			break;
		case (4) :
			goto RAMP_STAGE;
		case (5) :
			goto END;
		default :
			goto POLLING_STAGE;
	}


//Pause State--------------------------------------------------------------------------------
	PAUSE_STAGE:

		mTimer(20);

		if(((PIND&0x04)!=0x04)&&pause==0){
			BRK();
			LCDClear();

			LCDWriteStringXY(0,0, "C" );
			LCDWriteStringXY(0, 1, "S");

			LCDWriteStringXY(1,0, "B");
			LCDWriteIntXY(2, 0, num_CB, 2);
			LCDWriteIntXY(2, 1, num_SB,2 )
			LCDWriteStringXY(4,0, "A");
			LCDWriteIntXY(5, 0, num_CA, 2);
			LCDWriteIntXY(5, 1, num_SA, 2);
			LCDWriteStringXY(7,0, "S");
			LCDWriteIntXY(8, 0, num_CS, 2);
			LCDWriteIntXY(8, 1, num_SS, 2);
			LCDWriteStringXY(10,0, "W");
			LCDWriteIntXY(11, 0,  num_CW, 2);
			LCDWriteIntXY(11, 1, num_SW, 2);

			LCDWriteStringXY(14, 0, "B");
			LCDWriteStringXY(15, 0, "T");
			LCDWriteIntXY(14, 1, num_CT, 2);

			pause=1;
		}else if(((PIND&0x04)!=0x04)&&pause==1){
			LCDClear();
```

```
                        REV();
                        pause=0;
                        STATE=0;
                }

                STATE=0;
                goto POLLING_STAGE;

//RAMP STAGE-------------------------------------------------------------------------------------
        RAMP_STAGE:
        mTimer(20);

        if(((PIND&0x08)!=0x08)&&rampdown==0){

                LCDClear();

                rampdown=1;
        }else if(((PIND&0x08)!=0x08)&&rampdown==1){
                LCDClear();
                LCDWriteStringXY(0,0, "CANCEL RAMPDOWN");
                REV();
                rampdown=0;
                STATE=0;
        }

        STATE=0;
        goto POLLING_STAGE;

//Reflective State-------------------------------------------------------------------------------
        REFLECTIVE_STAGE:
                if(ADC_result_flag==1){ //if a reading has been completed and it is a min, set min
                        if(ADC_result<min_adc){
                                min_adc=ADC_result;

                        }
                        ADCSRA |= _BV(ADSC); //Make an ADC
                        ADC_result_flag = 0x00;  //clear the flag
                }
                if((PIND&0x02)==0x02){ //if the reflective sensor is still reading part, stay in state 2
                        STATE=2;
                        }else{ //if the part has passed, store the result in a linked list and reset
                                LCDWriteIntXY(8,0,min_adc,3);
                                initLink(&newLink);

                                if(min_adc<=Alum_MAX){
                                        newLink->e.itemCode=1;
                                        LCDWriteStringXY(0,0, "A");
                                        num_CA=num_CA+1;
                                        num_CT=num_CT+1;
                                }else if(min_adc>Steel_MIN && min_adc<=Steel_MAX){
                                        newLink->e.itemCode=2;
                                        LCDWriteStringXY(0,0, "S");
                                        num_CS=num_CS+1;
                                        num_CT=num_CT+1;
                                }else if(min_adc>White_MIN && min_adc<=White_MAX){
                                        newLink->e.itemCode=3;
                                        LCDWriteStringXY(0,0, "W");
                                        num_CW=num_CW+1;
                                        num_CT=num_CT+1;
```

```
                        }else if(min_adc>Black_MIN){
                                newLink->e.itemCode=4;
                                LCDWriteStringXY(0,0, "B");
                                num_CB=num_CB+1;
                                num_CT=num_CT+1;
                        }
                        if(head==NULL){ //If queue is broken, re-initialize.
                                setup(&head, &tail);
                        }
                        enqueue(&head, &tail, &newLink);
                        dispExit(&head);

                        STATE=0;
                        min_adc=0xfff;
                }
                goto POLLING_STAGE;
//Bucket Stage-----------------------------------------------------------------------------------------

        BUCKET_STAGE:
        BRK();

                        int current = StepPos; //get current position
                        int target = head->e.itemCode;   //Assigns target to const int val
                        int part_ID = target;
                        if(target == 1){
                                target=50;      //aluminum
                                }else if(target==2){
                                target = 150; //steel
                                }else if(target==3){
                                target=100; //White
                                }else if(target==4){
                                target=0; //black
                        }
                        //Determine Distance
                        int distance = current-target;
                        if(distance==150){
                                distance=-50;
                        }else if(distance==-150){
                                distance=50;
                        }
                        dequeue(&head, &rtnLink);


                        if(part_ID == 1){
                                //aluminum
                                --num_CT;
                                ++num_SA;
                                }else if(part_ID==2){
                                //steel
                                --num_CT;
                                ++num_SS;
                                }else if(part_ID==3){
                                //White
                                --num_CT;
                                ++num_SW;
                                }else if(part_ID==4){
                                //black
                                --num_CT;
                                ++num_SB;
```

```
                        }

                        mTimer(bucketDelay);
                        StepPos = stepperMover(distance, StepPos, Nintey_deg_Profile, One_Eighty_deg_Profile);
                        REV();



                STATE = 0;
                goto POLLING_STAGE;


//End State-------------------------------------------------------------------------------------------------------
        END:

        BRK();
        LCDClear();
        LCDWriteStringXY(0,1,"END");
        LCDWriteStringXY(1,0, "B");
        LCDWriteIntXY(2, 0, num_CB, 2);
        LCDWriteStringXY(4,0, "A");
        LCDWriteIntXY(5, 0, num_CA, 2);
        LCDWriteStringXY(7,0, "S");
        LCDWriteIntXY(8, 0, num_CS, 2);
        LCDWriteStringXY(10,0, "W");
        LCDWriteIntXY(11, 0,  num_CW, 2);
        return(0);

}
/*=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
=======================================END OF MAIN=============================================================
*/
//FUNCTIONS-------------------------------------------------------------------------------------------------------

//ISR FUNCTIONS---------------------------------------------------------------------------------------------------

ISR(INT0_vect){ //End Detector Interrupt
        STATE = 3;
}

ISR(INT1_vect){ //Triggered if reflective sensor is hit
        STATE = 2;
        ADCSRA |= _BV(ADSC); //Make an ADC
        PORTL=0b11111111; //turn on yellow and green lights
        cTimer=100; //Reset bTimer if part detected
}
```

```
ISR(INT2_vect){//pause functionality
        STATE = 1;
}

ISR(INT3_vect){//pause and finish functionality
        STATE = 4;
}

ISR(ADC_vect){ //Triggered upon and a completed ADC conversion
        ADC_result = ADC; //sets the ADC result when conversion complete
        ADC_result_flag = 1;     //sets the result flag

}



ISR(TIMER3_COMPA_vect){ //Triggered upon secondary timer completion

        cTimerInt = 1;

}


ISR(BADISR_vect)
{
        BRK();
        KILL();
        LCDClear();
        LCDWriteStringXY(0,0,"BAD ISR");

}

//STEPPER-----------------------------------------------------------------------------------------------------------------------

//INITIALIZE STEPPER MOTOR
// Input: Number of desired steps to be taken
//Returns: final step position
int initStepper(int n){
        //repeat for n cycles
        int stepOne = 0b00110000;
        int stepTwo = 0b00000110;
        int stepThree = 0b00101000;
        int stepFour = 0b00000101;
        int stepDelay = 15;

        for(int i=0; i<n; i=i+4){   //n is the desired number of steps
                PORTA = stepTwo;
                mTimer(stepDelay);
                PORTA = stepThree;
                mTimer(stepDelay);
                PORTA = stepFour;
                mTimer(stepDelay);
                PORTA = stepOne;
                mTimer(stepDelay);
        }

        while((PINH&0x01)==0x01){
                stepperCCW(1);
        }
```

```
                stepperCCW(7);  //+6 offset to account for end detector offset

        return 1; //return the step number
}

//CW
int stepperCW(int StepPos){
        //Define Stepper positions (From table)

        int stepOne = 0b00110000;
        //int stepOne = 0b00110101;
        int stepTwo = 0b00000110;
        //int stepTwo = 0b00111100;
        int stepThree = 0b00101000;
        //int stepThree = 0b00101110;
        int stepFour = 0b00000101;
        //int stepFour = 0b00100111;

        switch(stepIdx) {
                case 1:
                        PORTA = stepTwo;
                        stepIdx = 2;
                        mTimer(20);
                        break;
                case 2:
                        PORTA = stepThree;
                        stepIdx = 3;
                        mTimer(20);
                        break;
                case 3:
                        PORTA = stepFour;
                        stepIdx = 4;
                        mTimer(20);
                        break;
                case 4:
                        PORTA = stepOne;
                        stepIdx = 1;
                        mTimer(20);
                        break;
        }




        if(StepPos==199){
                StepPos=0;
        }else{
                ++StepPos;
        }


        return StepPos;
}


//CCW
int stepperCCW(int StepPos){
        //Define Stepper positions (From table)
```

```
        // 0 0 E0 L1 L2 E1 L3 L4

        int stepOne = 0b00110000;
        //int stepOne = 0b00101101;
        int stepTwo = 0b00000110;
        //int stepTwo = 0b00101110;
        int stepThree = 0b00101000;
        //int stepThree = 0b00110110;
        int stepFour = 0b00000101;
        //int stepFour = 0b00110101;
        int stepDelay = 15;

        switch(stepIdx) {
                case 1:
                PORTA = stepFour;
                stepIdx = 4;
                mTimer(stepDelay);
                break;
                case 2:
                PORTA = stepOne;
                stepIdx = 1;
                mTimer(stepDelay);
                break;
                case 3:
                PORTA = stepTwo;
                stepIdx = 2;
                mTimer(stepDelay);
                break;
                case 4:
                PORTA = stepThree;
                stepIdx = 3;
                mTimer(stepDelay);
                break;
        }

        if(StepPos==0){
                StepPos=199;
                }else{
                --StepPos;
        }

                return StepPos;
}

int stepperMover(int distance, int stepPos, int Nintey_deg_Profile[], int one_eighty_deg_Profile[]){

        //int stepOne = 0b00110000;
                int stepFour=0b00110110;
        //int stepTwo = 0b00000110;
                int stepThree=0b00110101;
        //int stepThree = 0b00101000;
                int stepTwo=0b00101101;
        //int stepFour = 0b00000101;
                int stepOne=0b00101110;

        if(distance==-50){

                stepPos=stepPos+50;
                if(stepPos>199){
```

```
        stepPos=0;
}

int i=0;
while(i<50){

        if(stepIdx==1){
                PORTA = stepTwo;
                stepIdx = 2;
                mTimer(Nintey_deg_Profile[i]);
                //mTimer(20);
                i++;

                }else if(stepIdx==2){
                PORTA = stepThree;
                stepIdx = 3;
                //mTimer(20);
                mTimer(Nintey_deg_Profile[i]);
                i++;

                }else if(stepIdx==3){
                PORTA = stepFour;
                stepIdx = 4;
                //mTimer(20);
                mTimer(Nintey_deg_Profile[i]);
                i++;

                }else if(stepIdx==4){
                PORTA = stepOne;
                stepIdx = 1;
                //mTimer(20);
                mTimer(Nintey_deg_Profile[i]);
                i++;
        }
}

}else if (distance==50){

stepPos=stepPos-50;
if(stepPos<0){
        stepPos=150;
}

int i=0;
while(i<50){

        if(stepIdx==1){
                PORTA = stepFour;
                stepIdx = 4;
                mTimer(Nintey_deg_Profile[i]);
                //mTimer(20);
                i++;

                }else if(stepIdx==2){
                PORTA = stepOne;
                stepIdx = 1;
                mTimer(Nintey_deg_Profile[i]);
                //mTimer(20);
                i++;
```

```
                }else if(stepIdx==3){
                PORTA = stepTwo;
                stepIdx = 2;
                mTimer(Nintey_deg_Profile[i]);
                //mTimer(20);
                i++;

                }else if(stepIdx==4){
                PORTA = stepThree;
                stepIdx = 3;
                mTimer(Nintey_deg_Profile[i]);
                //mTimer(20);
                i++;
        }
}

}else if (distance==-100){

stepPos=stepPos+100;
if(stepPos==200){
        stepPos=0;
        }else if(stepPos==250){
        stepPos=50;
}

int i=0;
while(i<100){

        if(stepIdx==1){
                PORTA = stepTwo;
                stepIdx = 2;
                mTimer(one_eighty_deg_Profile[i]);
                //mTimer(20);
                i++;

                }else if(stepIdx==2){
                PORTA = stepThree;
                stepIdx = 3;
                mTimer(one_eighty_deg_Profile[i]);
                //mTimer(20);
                i++;

                }else if(stepIdx==3){
                PORTA = stepFour;
                stepIdx = 4;
                mTimer(one_eighty_deg_Profile[i]);
                //mTimer(20);
                i++;

                }else if(stepIdx==4){
                PORTA = stepOne;
                stepIdx = 1;
                mTimer(one_eighty_deg_Profile[i]);
                //mTimer(20);
                i++;
        }
}
```

```
                }else if (distance==100){

                stepPos=stepPos-100;
                if(stepPos==-100){
                        stepPos=100;
                        }else if(stepPos==-50){
                        stepPos=150;
                }

                int i=0;
                while(i<100){

                        if(stepIdx==1){
                                PORTA = stepFour;
                                stepIdx = 4;
                                mTimer(one_eighty_deg_Profile[i]);
                                //mTimer(20);
                                i++;

                                }else if(stepIdx==2){
                                PORTA = stepOne;
                                stepIdx = 1;
                                mTimer(one_eighty_deg_Profile[i]);
                                //mTimer(20);
                                i++;

                                }else if(stepIdx==3){
                                PORTA = stepTwo;
                                stepIdx = 2;
                                mTimer(one_eighty_deg_Profile[i]);
                                //mTimer(20);
                                i++;

                                }else if(stepIdx==4){
                                PORTA = stepThree;
                                stepIdx = 3;
                                mTimer(one_eighty_deg_Profile[i]);
                                //mTimer(20);
                                i++;
                        }
                }
        }
        return stepPos;
}


//LINKED LIST FUNCTIONS-----------------------------------------------------------------

/*
* 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;
}

/*
```

```
* 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 **newLink){
        //link *l;
        *newLink = malloc(sizeof(link));
        (*newLink)->next = NULL;
        return;
}

/*
*  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;
}

/*
* DESC : Removes the link from the head of the list and assigns it to deQueuedLink
* INPUT: The head and tail pointers, and a ptr 'deQueuedLink'
*                 which the removed link will be assigned to
*/
/* This will remove the link and element within the link from the head of the queue */
void dequeue(link **h, link **deQueuedLink){
        /* ENTER YOUR CODE HERE */
        *deQueuedLink = *h;      // Will set to NULL if Head points to NULL
        /* Ensure it is not an empty queue */
        if (*h != NULL){
                *h = (*h)->next;
                free(*deQueuedLink);
        }/*if*/

        return;
}

/*
* 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);
```

```
}

/*
* 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;
}

/*
* 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);
}

/*
* 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);
}/*size*/

//DC MOTOR CONTROL FUNCTIONS-------------------------------------------------------------------------------
```

```
void BRK(){
        PORTB  = 0b00001111;
        mTimer(20);
}

void FWD(){
        PORTB  = 0b00001011;
}

void REV(){
        PORTB  = 0b00000111;
}

void KILL(){
        BRK();
        PORTB= 0b00001100;
}

//MTIMER-----------------------------------------------------------------------------------------------------
void mTimer(int count){
        int i;
        i=0;

        TCCR1B |=_BV(WGM12);

        OCR1A=0x03E8;

        TCNT1=0x0000;

        TIFR1 |= _BV(OCF1A);

        while(i<count){
                if((TIFR1 & 0x02)==0x02){
                        TIFR1 |=_BV(OCF1A);
                        i++;
                }
        }
        return;
}



//DISPEXIT: LCD Function for displaying the 1st 2 contents of linkedlist to LCD
void dispExit(link **head){
                switch((*head)->e.itemCode) { //Display First Loader
                        case 1:
                                LCDWriteStringXY(0,1, "A");
                                break;
                        case 2:
                                LCDWriteStringXY(0,1, "S");
                                break;
                        case 3:
                                LCDWriteStringXY(0,1, "W");
                                break;
                        case 4:
                                LCDWriteStringXY(0,1, "B");
                                break;
                }
                if((*head)->next != NULL){  //Print next up if there is one
```

```
                PORTL=0b11110000;
                switch((*head)->next->e.itemCode) { //Display Second Loader
                        case 1:
                                LCDWriteStringXY(2,1, "A");
                                break;
                        case 2:
                                LCDWriteStringXY(2,1, "S");
                                break;
                        case 3:
                                LCDWriteStringXY(2,1, "W");
                                break;
                        case 4:
                                LCDWriteStringXY(2,1, "B");
                                break;
                }
        }
}
```