University of Victoria

Mechanical Engineering

MECH 458

Mechatronics

Fall 2018

Automated Material Sorter
Final Project Report

Lab Section - B03

Group - 3

Prepared by:

Morgan Williams -

Submitted: January 6, 2019

Abstract

This report outlines the design and implementation process of creating the MECH 458 Automated Material Sorting lab project. As a result of this lab project, a fully functional material sorting system was realized. The system is controlled by a 8-bit AVR based microcontroller and comprises of a conveyor belt with an array of sensors that is driven by a brushed DC motor to feed a sorting tray that is mechanically coupled to a DC stepper motor for control. The system design was successful and bears testament to the robust design approach and implementation. The system benchmarked remarkably well, despite a lack of additional features that could have improved performance. Possible improvements include increased conveyor belt speed and sorting "hazard" management.

Contents

List of Figures

Listings

List of Tables

Executive summary

The designed system automatically sorts aluminum, steel, and black/white plastic materials using a 8-bit microcontroller to control a conveyor belt system driven by a brushed DC motor, as well as, a DC stepper motor to control the sorting tray. Sorting statistics are displayed on an LCD and the system also features buttons to both pause and ramp-down.

Testing and calibration was conducted using a variety of diagnostic tools and required to calibrate the ADC and the stepper motor "S-curve" movement profile. ADC calibration required recorded multiple iterations of item reflection measurements to generate a plot to determine thresholds to be used for item classification. Stepper "S-curve" calibration required testing a variety of dynamic loads in conjunction with modifying minimum step delay and acceleration/deceleration rates to achieve a movement profile that could output sufficient torque but also acceleration to be able to remove metal objects from the tray throughout sorting. The system performed remarkably well posting an system performance index of 1.243 (37 seconds/2 errors).

Performance limitations and trade-offs due to conveyor systems and sensor array configuration could be mitigated by developing a conveyor system with greater spacing between entrance and exit sensors. It is recommended that instead of testing by manually-loading a 48 item set, an automated loading is assumed and sorting is tested from belt rest to eliminate induced human loading error. The final design implementation successfully addressed all objects and were ultimately content with our lab experience as it offered plenty of learning experiences as well as reinforcement for a variety of electrical engineering concepts.

1 Introduction

1.1 Problem Statement

Recycling is a critical component of waste management. Many recycling methods are mature and efficient, the challenge derives from sorting materials efficiently. Material sorting is required because unique materials require unique recycling procedures. Currently, material sorting is primarily performed by manual labor. In an effort to improve sorting speed, our project aims to outperform a human counterpart. At a time when human waste production is seemingly unsustainable, developing an automated material sorting product is ultimately a critical component of meeting the demands of current and future waste management.

1.2 Design Objectives

The product is required to accomplish the following objectives:

- Accurately sort 4 unique materials into labeled tray.
- Accurately sort 48 items within 60 seconds.
- Provide user buttons to both pause and ramp-down system.
- Provide user with LCD interface for current sorting statistics.

1.3 Design Overview

The designed system automatically sorts aluminum, steel, and black/white plastic materials. For demonstration purposes, these materials are rounded objects that are fed to the sensor array via conveyor belt. The conveyor belt is driven by a brushed DC motor and feeds a sorting tray that is controlled by a DC stepper motor. All sensors and motors are controlled via the AT90 USB1287 microcontroller (MCU). Sorting begins when the sensor array detects and subsequently classifies objects and generates a queue. When the object is detected at the end of the conveyor belt by the exit optical sensor, the belt is stopped and the stepper motor is signaled to the corresponding sorting position using the queue information. When the stepper position is validated, the belt is engaged to pass the part into the sorting tray.

1.3.1 AT90 USB1287 MCU

The Atmel AT90 USB1287 is an 8-bit AVR based microcontroller (MCU) powered by a 3.3V supply. The 3.3V supply forces logic operation to the same voltage. The default system core clock is 1 MHz but has been maxed to 8 MHz for the purposes of this project. This MCU is a system-on-chip that includes a variety of other systems, most notably a 10-bit ADC and PWM. The embedded system design will leverage these devices in order to complete a variety of tasks. A complete electrical hardware schematic is provided in the Appendix A.

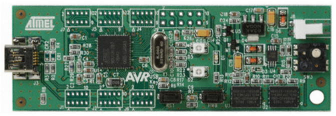


Figure 1: Atmel AT90 USB1287 development board.

1.3.2 Conveyor and Brushed DC Motor

The conveyor is driven by the brushed DC motor and is outfitted with guides to ensure objects travel to the sensors with correct spacing. The Brushed DC motor is responsible for controlling the conveyor belt speed. The motor's rotor is mechanically coupled directly to a gearbox (for reduction) and the conveyor belt as shown in Figure 2. The ST VNH3SP30 IC in Pololu's MD01B breakout board is used to drive the Brushed DC motor and is controlled via AT90 USB1287 GPIO.

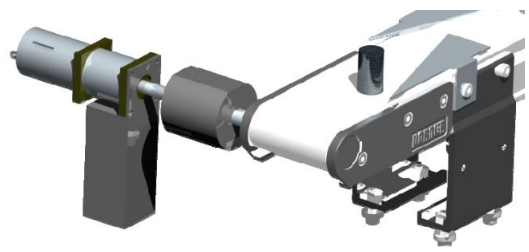
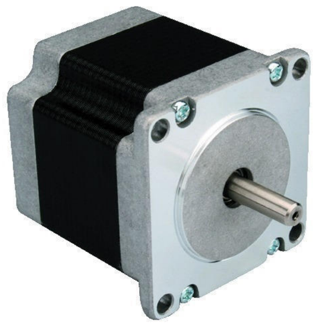


Figure 2: Brushed DC motor (left) and VNH2PSP30 driver IC breakout board (right).

1.3.3 DC Stepper Motor

The DC stepper motor rotor is directly coupled to the sorting tray to enable to control the sorting tray position with respect to the exit of the conveyor belt. The L298 stepper motor driver IC integrated within a breakout board is used to drive the DC stepper motor. AT90 USB1287 GPIO is used to control the L298 IC.



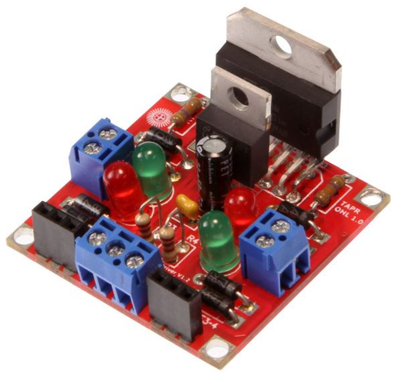


Figure 3: DC stepper motor (left) and L298 driver IC breakout board (right).

2 System Design

The following section outlines the hardware and software co-design algorithm that includes the Finite State Machine, tray sorting, DC stepper acceleration/deceleration, conveyor drive system, conveyor sensor array, and system control interface.

2.1 Finite State Machine

The Finite State Machine (FSM) connects the separate blocks of code together and enables the system to seamlessly transition between functional blocks of code. By default the system will always return to the "POLLING" state as shown in Figure 4.

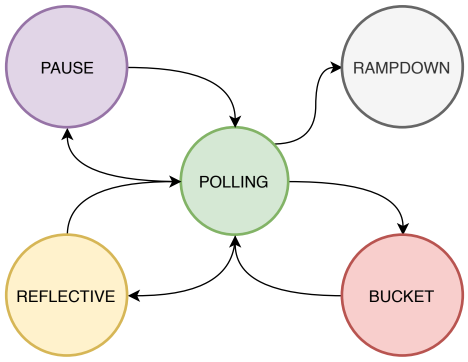


Figure 4: Finite state machine diagram.

The current state is stored in a global variable "STATE". "STATE" is updated accordingly to transition between states. A switch case is implemented to structure the FSM logic as shown in Listing 1.

Listing 1: Finite state machine logic.

```
STATE = POLLING; // Default state
switch(STATE){
        case(POLLING):
                STATE = POLLING;
                break;
        case(REFLECTIVE):
                STATE = POLLING;
                break;
        case(BUCKET):
                STATE = POLLING;
                break;
        case(PAUSE):
                if ((pauseFlag==1))     {
                        LCDWriteStatistics(); // Display statistics
                }
                while(pauseFlag==1); // Do nothing until flag deasserted
                break;
        case(RAMPDOWN):
                cli(); // Disable global interrupts.
                LCDWriteStatistics();
                STATE = POLLING;
                break;
        default:
                STATE = POLLING;
}
```

2.1.1 Polling State

"POLLING" is analogous to "idle" of the state machine. Inside this state, the system waits to service interrupts from either pause, ramp-down, and or exit and entrance optical sensors.

2.1.2 Reflective State

The"REFLECTIVE" state is activated when the entrance proximity laser senor (OR) located near the reflective sensor detects an object as shown in the code below. This interrupt service routine starts an ADC conversion to begin the object classification sequence. This operation is illustrated in Listing 2. Following object classification, the value is stored in the sorting queue and the state is set to "POLLING".

Listing 2: Entrance optical sensor (OR) interrupt service routine.

```
ISR(INT1_vect){
        STATE = REFLECTIVE;
        ADC_result = 0xFFFF;// Set to max value
        ADCSRA |= _BV(ADSC);// Start ADC conversion
}
```

2.1.3 Bucket State

The "BUCKET" state is activated when the exit sensor (EX) is triggered by object obstruction. In this state the belt is stopped and the sorting tray turns to the correct position, sorting statistics are updated, the sorted item is removed from the queue, and the belt is driven to drop the item into the sorting tray. This operation is shown in Listing 3.

Listing 3: BUCKET stage interrupt service routine.

```
ISR(INT0_vect)  {
        dcDrive(BRK, 0); // Stop conveyor
        stepperStateChange(front(&q)); // Move stepper to head item
        switch(front(&q)){ // Sorting statistics update
                case ALUMINUM:
                        AL_count++;
                        break;
                case BLACK:
                        BL_count++;
                        break;
                case WHITE:
                        WH_count++;
                        break;
                case STEEL:
                        ST_count++;
                        break;
                default:
                        break;
        }
        pop(&q); // Remove sorted item from queue
        dcDrive(DC_CW, BELT_DUTY); // Put item into bucket
        STATE = BUCKET;
}
```

2.1.4 Pause State

The “PAUSE” state is activated the pause button is toggled. This button is configured with as an interrupt and triggers the pause ISR. The pause ISR asserts a ”pauseFlag” and stops or starts the conveyor belt. This functionality is illustrated below in Listing 4.

Listing 4: PAUSE interrupt serivce routine.

```
ISR(INT4_vect)  {

        dcDrive(BRK, 0);// Stop belt

        if (pauseFlag==0){ // Toggle pause flag
                pauseFlag = 1;
                STATE = PAUSE; // Update state
        } else {
        pauseFlag = 0;
        dcDrive(DC_CW,BELT_DUTY);// Start belt
        STATE = POLLING;// Set back to polling state
        }
        msTimer(20);// De-bounce switch
}
```

When the system is paused, diagnostic information and sorting statistics is displayed on the LCD as discussed in Section 3.1.2. A short 20 ms timer is used to “debounce” the pause button. Mechanical switch “bounce” from the mechanical switch contacts and manifest as an oscillating electrical signal. By installing this timer, the oscillation period is circumvented.

2.2 Sorting Queue

The function of the sorting queue is to store object classification data as well as the length. The custom sorting queue implementation consists of a global structure Queue that consists of elements head and size. The queue is composed of node elements that contain item description as well as a pointer to the next element. From this singly-linked list structure, queue functionality is implemented. Queue commands push and pop are included in Listing 5. A visualization of the queue commands are included in Figure 5.

Listing 5: PAUSE interrupt serivce routine.

```
void push(struct Queue *q, int data) {
        q->size++; // Increment queue size
        struct node *current = q->head;
        if (q->head == NULL) { // If queue empty
                q->head = (struct node *) malloc(sizeof(struct node));
                q->head->item = data;
                q->head->next = NULL;
                return;
        }
        else {
                while (current->next != NULL){ // Fin
                        current = current->next;
                }
                current->next = (struct node *) malloc(sizeof(struct
                    node));
                current->next->item = data;
                current->next->next = NULL;
        }
}

void pop(struct Queue *q) {
        if (q->size == 0){ // If queue empty
                q->size = 0;
                return;
        }
        else {
                q->size--; // Decrement size
                struct node *tmp = q->head;
                q->head = q->head->next; // New head
                free(tmp); // Free memory allocated to old head
        }
}
```

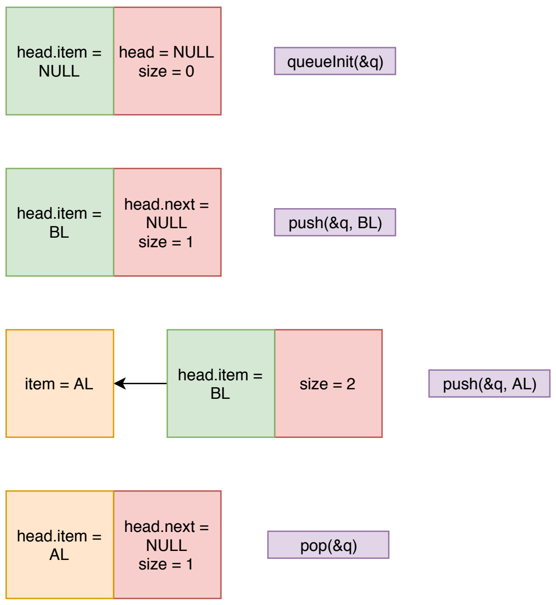


Figure 5: Sorting queue functional diagram.

2.3 Brushed DC Motor Control

The conveyor belt is driven by a mechanically coupled brushed DC motor seen in Figure 2. Control of the brushed DC motor is achieved by using AT90 GPIO commands and PWM to interface with the ST VNH3SP30 IC. A level-shifting IC is required to control the L298N via our MCU because its operates at 5V, where as the MCU GPIO operates at 3.3V. AT90 GPIO are set to command the VNH3SP30 to configure DC motor polarity for clockwise rotation. PWM frequency is set to 600 Hz (though the IC can handle much higher) and the calibrated duty to 25%. DC motor voltage is proportionate to PWM duty with a voltage-speed control implementation. Equation 1 illustrates this relationship used to control DC motor speed. A detailed hardware schematic is included in Appendix A.

$$V_{DC-out} = Duty(\%) \times V_{DC-in} \tag{1}$$
$$Speed(rpm) \propto V_{DC} \tag{2}$$

2.4 DC Stepper Motor Control

The DC stepper motor was designed to be as fast as possible, while outputting sufficient torque to remove metallic pieces from the bucket during angular accelerations. The removal of these pieces is critical during extended use as the weight of sorted pieces accumulates and reduces stepper speed. In order to realize these goals, the following stepper control techniques were employed.

2.4.1 Full-Stepping Sequence

The full-step sequence was used to improve the average torque output. Despite the lost resolution of full-stepping [1], experimentation revealed that this was insignificant and the improved average torque output was more desirable for this application. Figure 6 illustrates the full-step sequence and that the increased average torque derives from the higher average coil current.

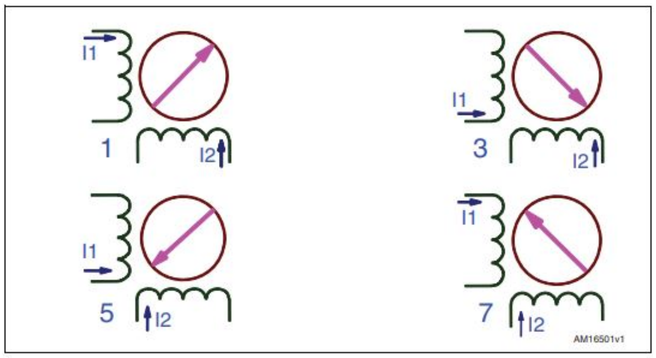


Figure 6: Full-step sequence for a two-phase bipolar motor [1].

2.4.2 Acceleration and Deceleration Methodology

The DC stepper motor acceleration and deceleration profile was created by incorporating a number of industry and academic inspirations. First, we referenced [2] to form the acceleration and deceleration profiles with respect to the entire curve. It was determined that 20% of steps equally dedicated to the acceleration and deceleration respectively. A 90 degree turn of 50 steps would consequently require the first and last 10 steps to be dedicated to acceleration and deceleration. This relationship is illustrated in Figure 7.

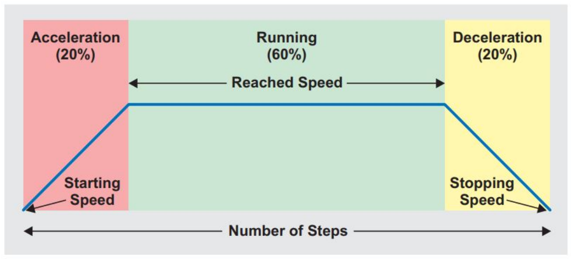


Figure 7: DC stepper motor speed curve [2].

The "S-curve" technique was used for acceleration and deceleration curves to reduce "motor jitter" observed at the vertices of the profile. "S-curve" derives from the Sigmoid function shown in Figure 8. In order to derive the function required to calculate the S-curve functions, the "Logistic Function" [3] was used as a reference point. From the logistic function seen in Equation 3, we derive the acceleration and deceleration functions by manipulating variables L, k, and x_0 within the respective domains of the acceleration and deceleration regions. L corresponds to the minimum delay between stepper sequence pulses, k corresponds to the slope of the "S-curve", and x_0 to the midpoint [3].

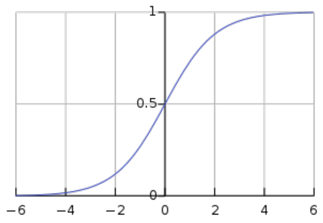


Figure 8: Sigmoid function.

$$f(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{3}$$

$$90^\circ\ Acceleration \rightarrow delay = \frac{(Min.\ Delay)}{1 + e^{-k(x-5)}} + (Min.\ Delay) \tag{4}$$

$$90^\circ\ Deceleration \rightarrow delay = \frac{-(Min.\ Delay)}{1 + e^{-k(x-45)}} + 2(Min.\ Delay) \tag{5}$$

$$180^\circ\ Acceleration \rightarrow delay = \frac{(Min.\ Delay)}{1 + e^{-k(x-10)}} + (Min.\ Delay) \tag{6}$$

$$180^\circ\ Deceleration \rightarrow delay = \frac{-(Min.\ Delay)}{1 + e^{-k(x-90)}} + 2(Min.\ Delay) \tag{7}$$

2.4.3 Sorting Tray Algorithm

Mechanical coupling of the DC stepper motor to the sorting tray enables tray state transition control. The stepper motor used has a step resolution of 1.8°. From this relationship we derive that 90° and 180° turns correspond to 50 and 100 steps. State transitions for clockwise and counter-clockwise turns manifest as tray state transitions.

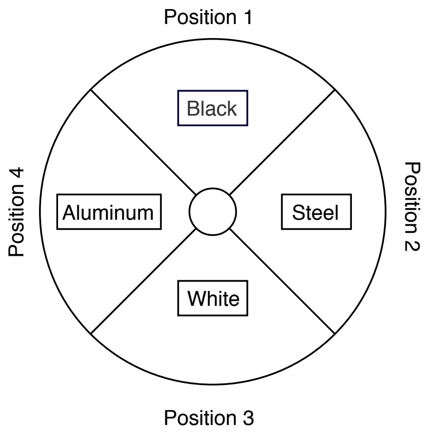


Figure 9: Stepper tray state diagram.

Figure 9 illustrates the numbered tray states used to derive the optimized stepper state transition function. The stepper state transition function computes the distance vector between destination and origin states and uses this value to determine the required stepper action to reach the destination state. Listing 6 illustrates the relationship used to compute tray state transitions with respect to the DC stepper motor.

Listing 6: Stepper tray transition function

```
void stepperStateChange(int destination){
    int cmp = destination - stepperState;
    if(cmp ! = 0){ // If destination != origin
        if( (cmp == -1) || (cmp == 3) ){
            stepperRotateCW90(); // Rotate CW 90 degrees
            stepperState = destination; // Update stepper state
            return; // Exit
        }
        if( (cmp == 1) || (cmp == -3) ){
            stepperRotateCCW90(); // Rotate CCW 90 degrees
            stepperState = destination;
            return;
        }
        if( (cmp == 2) || (cmp == -2) ){
            stepperRotateCW180(); // Rotate CW 180 degrees
            stepperState = destination;
            return;
        }
    }
    stepperState = destination;
}
```

2.5 Reflective Sensor

The ADC is triggered by the REFLECTIVE state and reads the output voltage of the reflective sensor. The reflective sensor measures reflected content from its light emitting element. This measurement is output with respect to its supply voltage (3.3V). The larger output voltage measurements indicate low reflectivity (close to 3.3V) and lower measurements (closer to 0.6V) indicate high reflectivity. The inversely proportionate relationship derives from the photo-transistor topology of the reflectivity sensor. Increased reflected light results in larger photo-transistor bias current, reducing output voltage. Due to the curvature of the objects, accurate readings are taken near the center of the objects as they pass the reflective sensor. Readings taken at the center of objects will be the minimum values because this is the optimal angle of incidence for the reflective sensor. To ensure the most valid readings are taken, the ADC interrupt service routine continually triggers itself (somewhat recursively) while the object is within the entrance optical sensor (OR) in order to capture the minimum reading. This minimum reading is then used to classify the object type by cross-referencing the ADC calibration. The classified object is added to the queue and the system returns to the idle state. This functionality is shown below in Listing 7.

The Reflective sensor output is connected to the 10-bit Analog to Digital Converter (ADC). 10-bit resolution is required to distinguish between white and black pieces. An AVR library variable “ADC” stores the 10-bit ADC output and does not require special care that is typically required with an 8-bit system.

Listing 7: Reflective sensor reading function.

```
ISR(ADC_vect){
        if (ADC < ADC_result){ //check for minimum value
        ADC_result = ADC;
        }
        if ((PIND&OR)== OR){ // Check if OR is high (piece is still in
            sensor)
        ADCSRA |= _BV(ADSC);
        }
        else { // Piece is out of sensor, begin classification...
                if (ADC_result < ALUMINUM_MAX_ADC){
                Object_result = ALUMINUM;
                }
                else if (ADC_result < STEEL_MAX_ADC){
                Object_result = STEEL;
                }
                else if (ADC_result < WHITE_MAX_ADC){
                Object_result = WHITE;
                }
                else if (ADC_result < BLACK_MAX_ADC){
                Object_result = BLACK;
                }
                else {
                Object_result= UNKNOWN;
                }
        push(&q, Object_result); // Push result to queue
        STATE = POLLING; // Return to polling state
        }
}
```

2.6 Laser Proximity Sensors

The conveyor belt has Laser Proximity sensors installed at the locations seen in Figure 10. Laser Proximity sensors consist of photo-transistor that will trigger a falling edge (EX) or a rising edge (OR) when its laser is obstructed. Laser obstruction constitutes object detection that is used to transition the Finite State Machine.

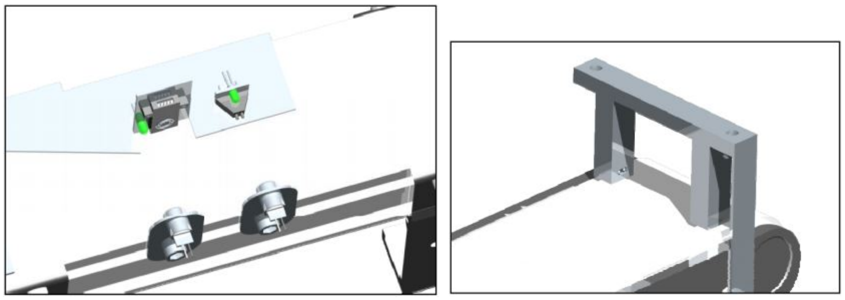


Figure 10: Conveyor belt optical sensors.

2.7 System Control Interface

2.7.1 LCD Interface

The LCD interface as shown in Figure 11 is used as a Human-Machine Interface (HMI) to enable system calibration and sorting tray data. Appendix B includes the LCD schematic. The generic 1602A 16X2 character LCD was chosen because of its ability to its 3.3V compatibility (for communications, requires 5V supply) and the ability to operate using 4-bit communication. Half-byte communication is critical because GPIO port pins are limited in this design. Additionally, writing custom display drivers are not required as they are readily available.

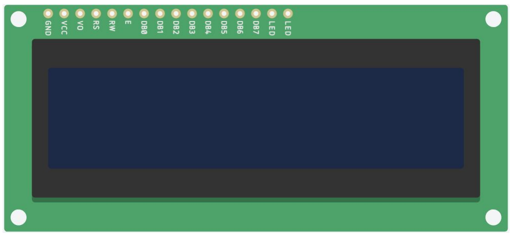


Figure 11: 1602A 16x2 character LCD.

2.7.2 System Pause Button

The System Pause button allows the user to halt/resume system operation and view current system statistics at any time. This button is in a pull-up configuration (active low) and configured as an interrupt. The pause interrupt service routine is triggered on initial press and pauses the entire system operation. The pause interrupt service routine display current sorting count by object as well as the current number of items on the conveyor. Another button press is required to exit the pause interrupt service routine to resume operation.

2.7.3 System Ramp Down

The System Ramp Down allows the user to sort the remaining the conveyor belt objects and automatically shut down and display the final sorting statistics. This functionality is achieved using the ramp down button and timer interrupt. System ramp down is engaged by a button press and triggers an interrupt timer that will complete only if the sorting queue is empty. Queued objects read during this timer countdown restart the timer, effectively suspending the system shut down until the conveyor belt objects have completed sorting. The timer interrupt threshold value must be configured to be large enough to allow an object at the end of the be detected and also large enough to sort the queued object. Experimentation found that this timer interrupt threshold value should be at least 3 seconds.

3 Testing and Calibration

The following section details to methods and procedures used to test and calibrate the various system components in order to achieve stable and repeatable output.

3.1 Diagnostic Systems

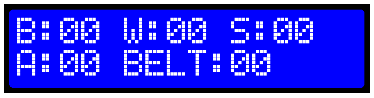
The following methods and procedures used to troubleshoot and diagnose the system component functionality.

3.1.1 Onboard LED Diagnostics

Onboard LEDs were used to convey quick diagnostic information. This technique is useful for real-time programming because diagnostic techniques like printing outputs are too cumbersome and consume too many resources for real-time applications. Instead of writing explicit GPIO commands, a custom driver was written. This custom driver is contained as header file (and method .c file) "diagnostic.h" in the Appendix.

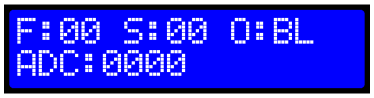
3.1.2 LCD Diagnostics

Diagnostic values such as sorting statistics (B: = black, W: = white, S: = steel, A: = aluminum, and BELT: = conveyor item count), queue information (F: = front and S: = size), object classification (O:), and ADC value (ADC:). Figures 12 and 13 illustrate the display outputs.



```
B:00 W:00 S:00
A:00 BELT:00
```

Figure 12: Sorting and belt statistics diagnostic display.



```
F:00 S:00 O:BL
ADC:0000
```

Figure 13: Queue, classification, and ADC diagnostic display.

3.2 Stepper S-Curve Generation and Calibration

Stepper motor acceleration is achieved by using modified delays between steps, as discussed in Section 2.4.2. MS Excel was used to construct custom delay arrays for 90 ° and 180 ° rotations. This curve is calculated using the equations found in Section 2.4.2 and allows the modification of individual S-curve parameters like the acceleration/deceleration speed and minimum delay. The process of calibrating the stepper motor involved modifying these S-curve parameters and testing them with respect to a number of different loads. The S-curve parameters selected compromised on maximum speed in order to ensure stable and high torque rotations. High torque output is important as the load is dynamic and requires the stepper to be able to "eject" metal parts buring rotations to lower the total load throughout operation. Figures 14 and 15 illustrate the final stepper S-curve calibration. It is important to note that at least microsecond delay precision is required to achieve accurate S-curve function.

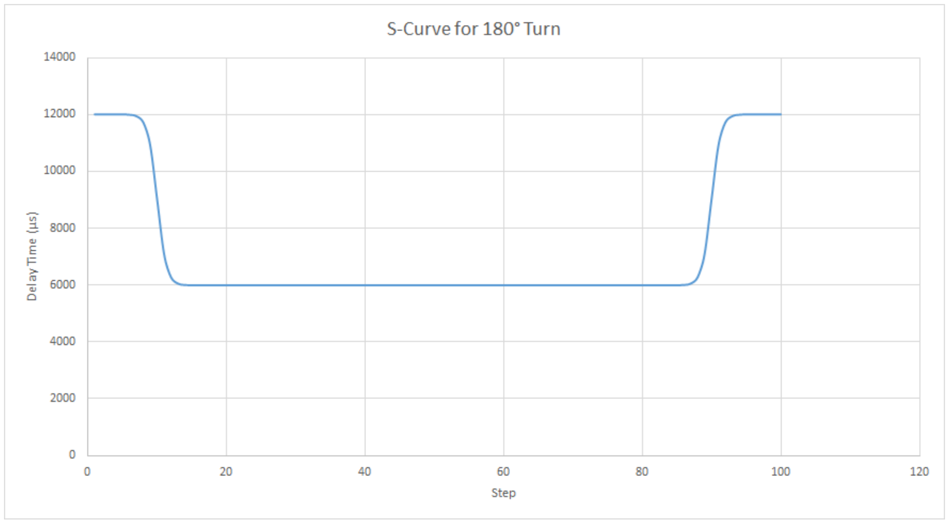


Figure 14: S-Curve for 180 ° turn: L = 6 ms and k = 1.5

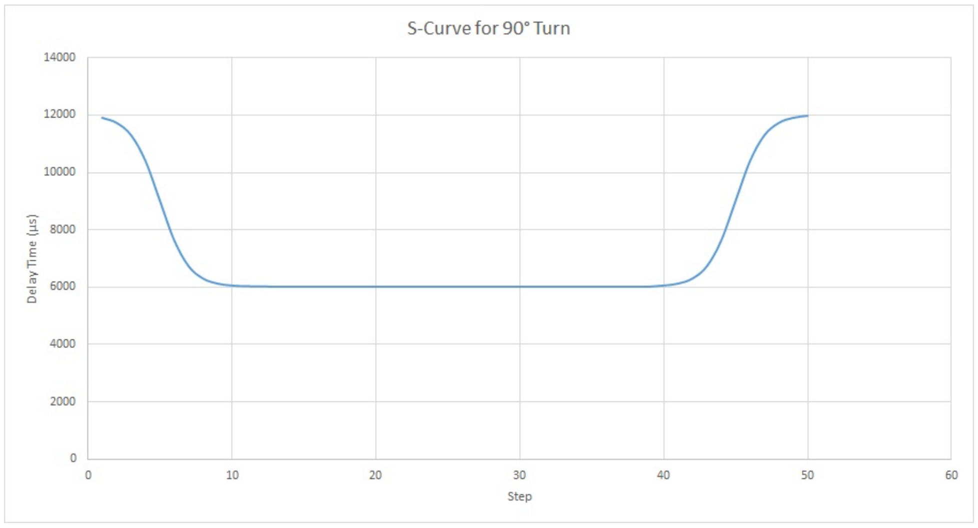


Figure 15: S-Curve for 90 ° turn: L = 6 ms and k = 1.5

3.3 ADC Calibration

ADC calibration was conducted by recording and plotting the LCD diagnostic display values of multiple test objects. We found the reflective properties of the aluminum and steel objects made them easiest to distinguish. This revelation is easily shown in Table 1. White and black objects presented challenges as their readings were close and also subject to the surface cleanliness of the white parts. In some cases, the ADC output for black objects would overlap with readings from white objects as shown in Figure 16. After running several trials, thresholds shown in Table 1 were derived and resulted in occasional object identification errors.

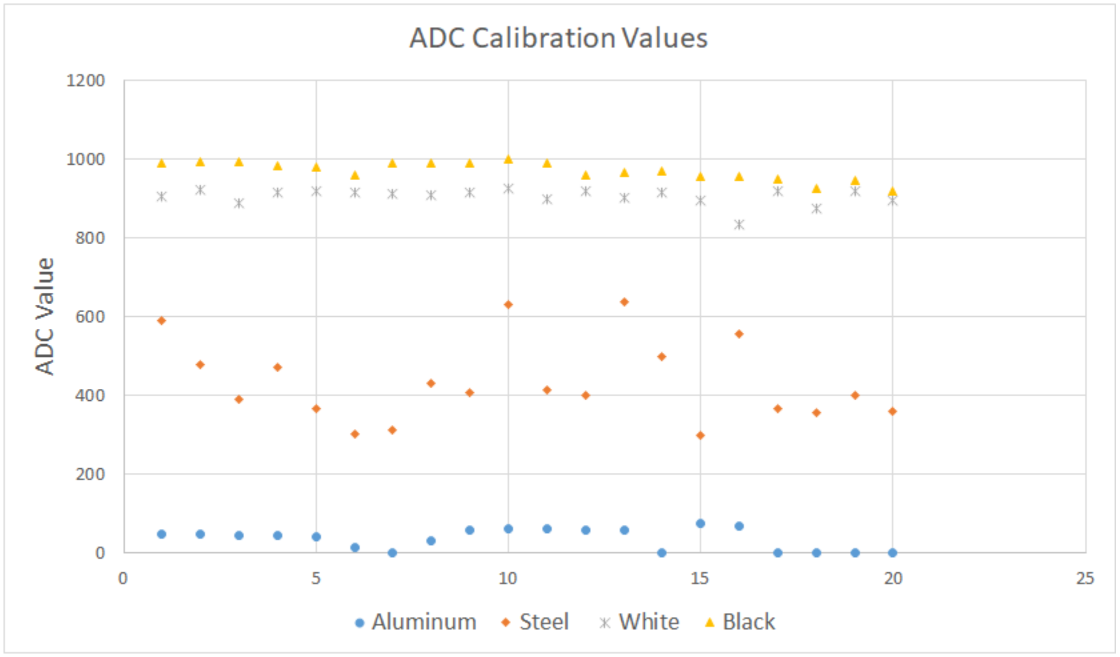


Figure 16: ADC calibration values for aluminum, steel, white, and black objects.

	Aluminum	**Steel**	**White**	**Black**
Minimum	0	300	834	920
Maximum	77	638	924	998
Average	36	434	904	970
Threshold	200	700	921	1024

Table 1: ADC calibration minimum, maximum, average, and threshold values.

4 Results

The following section discusses the final results of comprehensive system testing.

4.1 System Operation

Optimal system operation requires manually loading a batches of 8 pieces at the start of the belt sequence and continually adding 8 piece batches as the previous batch begins sorting at the exit optical sensor. This "batching" method can be performed with a lower count of items. Initial testing used 4 and 6 pieces. It was found that spacing between batch items did not effect system performance. 8 piece batches was used to improve sorting time. The batch sequence is signaled by the halted conveyor belt, a function of the sorting tray algorithm to ensure the sorting tray is in a valid position. This spacing between the batches of 8 pieces is required due to the distances between the exit and entrance optical sensor. In our final edition of our code, the case where both sensors are triggered is not handled. This issue is discussed in Section 5.

4.2 System Performance

System performance is evaluated using the "System Performance Index" shown in Equation 8, where N_C is the total number of correctly sorted objects, N_I is the total number of incorrectly sorted objects, and T_{MAX} is the time required to sort all 48 pieces. The two best test results are shown in Table 2 with their corresponding SPI value.

$$SPI = \frac{N_C - N_I}{T_{MAX}} \tag{8}$$

Time (s)	**Errors**	**SPI**
41	0	1.171
37	2	1.243

Table 2: System testing results.

5 System Performance Limitations and Trade-offs

Object loading limitations derive from the length of the conveyor belt and variability of manual loading. These limitations are inherited from the equipment and cannot be altered. The current configuration only allows for a maximum of 8 piece batch loading. This limitation comes from the fact that this length of items ensures that the entrance (OR) and exit (EX) optical sensor are not triggered simultaneously. Software is able resolve this conflict but our group did not have enough time to implement a proper solution. Object loading limitations, with respect to the demo, are bounded by the loader's ability. Variability introduced by manual loading could easily be mitigated by automated loading. Increasing conveyor belt length, automating object loading, and increasing sensor spacing, would drastically improve system performance. This performance limitation ostensibly proves that our code was not a bottleneck. This excellent outcome is a result of the increased clock speed optimization of each function.

Stepper motor speed is bounded by the unique physical parameters of the stepper motor. The motors used are sufficient but other models could provide more torque and speed that would improve performance. Stepper sorting algorithms are also bounded by the nature of our single-threaded control system. Our system can effectively only do one thing at a time. By giving our system components autonomy, our system performance could dramatically improve. Giving components their own processing unit allows for paralleled work flow but requires a communication network to transfer information between nodes. Controller Access Networks (CAN) and its many iterations are a popular technique to create a network of functional nodes that can communicate between each other.

Conveyor belt speed is a limitation with respect to manual loading and the braking distance. Manual loading is made increasingly difficult with faster conveyor belt speeds but increased speeds reduces sorting times. Increased speeds also increases braking distance. Braking distance is critical when sorting closely spaced objects, as a brake distance that exceeds the object spacing will result in an erroneous sort. Conveyor belt speed was reduced in order to allow for easier object loading and improved braking performance.

Items enter the sorting tray by falling off the end of the conveyor belt. This interface is at the mercy of the speed at which the object leaves the conveyor belt and the time required for the object to fall into the sorting tray. A part moving too quickly off the conveyor will overshoot the sorting tray. Sorting between objects must respect the time required for the object to fall into the sorting tray. If the time between sorting objects is shorter than the elapsed fall time, the sorting tray will preempt an incorrect sorting tray transition.

Given equipment imposed restrictions, few design trade-offs were made. Constant object loading is not possible in this setup so it cannot be said the batch loading was a compromise - it was a necessity. Constant loading in our setup is erroneous as our single threaded control unit cannot handle entrance (OR) and exit (EX) optical sensor being simultaneous triggering. Unique control units for the reflective and entrance optical sensor and for the stepper, conveyor belt drive, and exit sensors could be used to mitigate this issue. As stated earlier, this would require a robust and fast communication network architecture.

6 Experience and Recommendations

The conveyor system was fixed but could be improved by increasing the belt length, automating object loading, and increasing sensor spacing as discussed in Section 5. Manual loading in our setup was erroneous because of human input error and could be removed by an automated loading mechanism. Unique control units for the reflective and entrance optical sensor and for the stepper, conveyor belt drive, and exit sensors could be used to mitigate this issue. As stated earlier, this would require a robust and fast communication network architecture.

The overall lab experience was positive but could have been improved particularly with respect to the project performance evaluation. We propose that instead of using manual loading as a metric for sorting speed and accuracy, an automated loading system is assumed and that the conveyor starts sorting from rest. With this assumption, sorting could be triggered by an external interrupt vector (could be an additional button for demonstration) and would completely eliminate evaluating system performance based on manual loading proficiency. This loading assumptions would also expedite testing and the overall design would benefit from better power efficiency by only being triggered by an automated loader.

7 Conclusion

The final design implementation both accomplishes and exceeds all objectives. We are pleased with the addition of the LCD diagnostic display as well as the refined stepper acceleration and deceleration profiles. System performance was excellent given conveyor belt loading constraints and software was fast enough to not impede system operation. Our highest SPI run (37 seconds)/(2 errors) exceeded our anticipated results. System performance could have been improved by implementing sorting hazards algorithms to run our system without batched item input. The overall lab experience was positive but could have be improved with respect to the project performance evaluation. We propose that instead of using manual loading, an automated loading system is assumed and that the conveyor starts sorting from rest to eliminate human loading error.

References

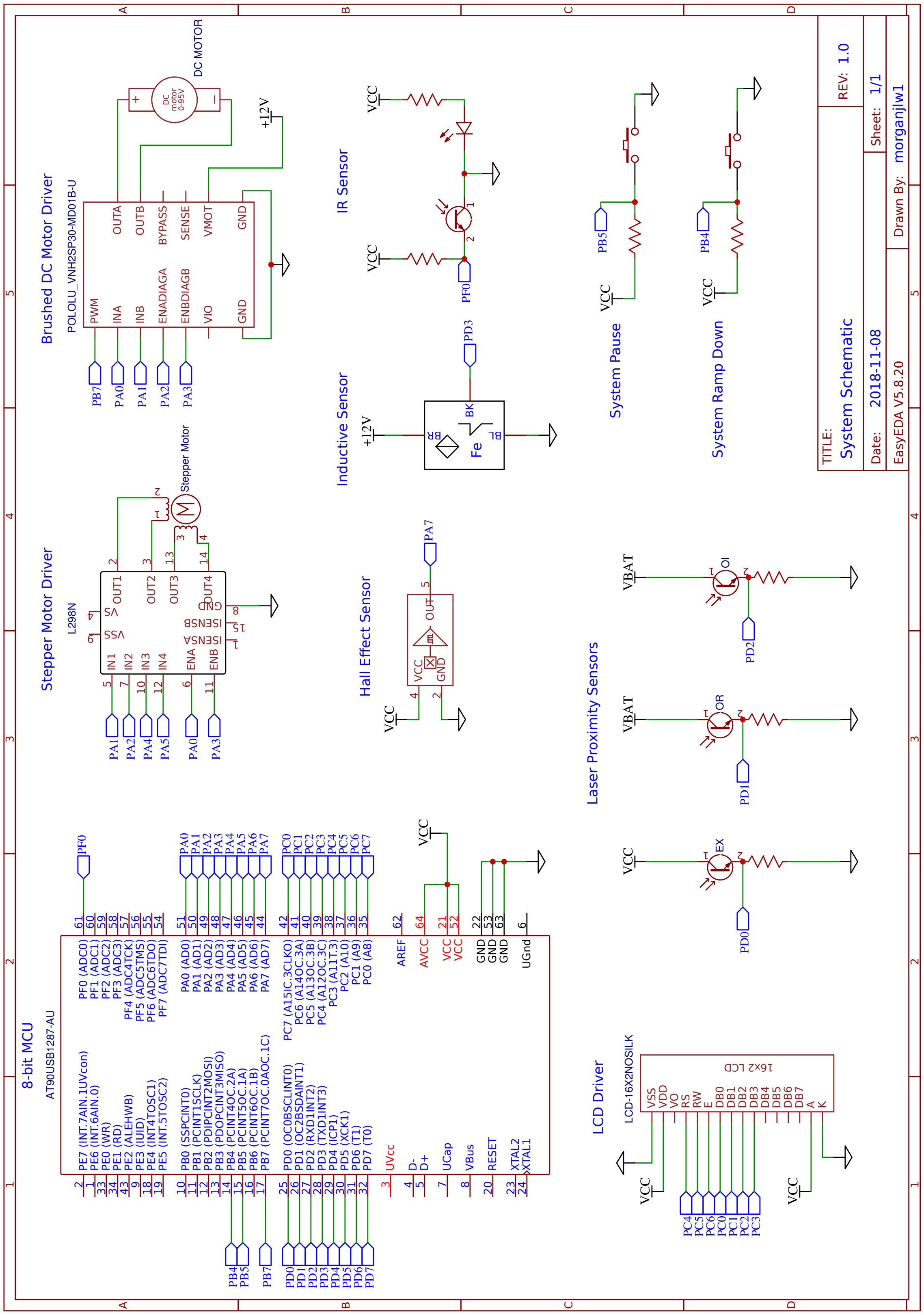
[1] "AN235 Application Note - Stepper motor driving", ST Electronics, Thomas Hopkins (2012).

[2] "Applying acceleration and deceleration profiles to bipolar stepper motors", Texas Instruments, Jose I. Quinones (2012).

[3] "Logistic Function", Wikipedia, https://en.wikipedia.org/wiki/Logistic_function.

Appendices

Appendix A - Hardware Schematic



Appendix B - Code

Find all source code attached separately.