Mech 458

Project Final Design Report

Reilly Reilly V00894910
Ryland Nezil V00157326

Executive Summary

The purpose of this project is to design a mechatronic system capable of sorting 48 uniformly shaped pieces comprising 4 different materials into separate bins. The bins will be mounted to the top of a stepper motor, and the materials consist of steel, aluminium, black plastic and white plastic. Materials must be identified via reflective sensor. Items are to be transported to the bins via conveyor belt, which in turn will be driven by a brushed DC motor. The reflective sensor shall be mounted in the middle of the conveyor belt, as will an optical sensor which can be used for motion detection. Another optical sensor will be mounted at the end of the conveyor belt. It is required that the sorting be completed in under 60 seconds with minimal errors. The system is to be controlled by an ATmega2560 microcontroller, which must be programmed in C. Auxiliary controls must include a killswitch, a pause/resume button, and a ramp down button. The ramp down button must allow the system to sort any items that are currently on the belt before shutting itself down. The system must be equipped with an LCD screen that will output information regarding the status of the system, the number of items sorted, and the number of items currently on the belt. Design, implementation and demonstration of this project were completed successfully, though the performance of the system was relatively poor. This was due to lack of planning with respect to interactions between system components, which necessitated a choice between reliability and performance. If performance had been chosen, the loss of reliability would have made the system fail to meet its requirements; therefore, reliability was chosen at the cost of system performance.

Abstract

The sorting of 48 pieces of four different colors and three different material types with one error or less as fast as possible, mandatory under 60 seconds, was successfully completed with the successful tests completing in 51 seconds with no errors and 45 seconds with 1 error. This was accomplished by programming an Atmega 2650 microcontroller to control input and output ports and managing interrupts to control the hardware devices of the apparatus. Input pins of the microcontroller were set to read data from a reflectivity sensor which an analog to digital converter would interpret to determine the color and material of pieces. Output pins were set to send data to control the stepper motor and DC motor and to the LCD to display the sorting results. External interrupts were used to manage the homing sensor of the sorting disk, exit sensor at the end of the conveyor belt, optic sensor paired with the reflectivity sensor, pause button, ramp down button and a killswitch. An internal interrupt was used for the analog to digital converter. The interrupts were managed to ensure proper system control and data acquisition. Timers were used to create delays in the code to allow time for physical manifestations of code to present, ensure all parts were cleared from the conveyor belt before

shutdown and to dictate the amount of time the analog to digital converter should take readings for a piece. Some issues presented during final testing when running pieces in close succession due to double counting at the exit sensor despite using a debouncing circuit to avoid pulse spikes from the DC motor from triggering the sensor. Further investigation is needed to determine methods to rectify this issue.

Acknowledgements

We would like to thank Patrick Chang, Dr. Yang Shi, Yue Song, Binyan Xu and Tianxiang Lu for all their advice and assistance with this project in the lab. Without their knowledge, experience and support, this project would not have been successful.

Table of Contents

List of Figures

List of Tables

1.0 Introduction

1.1 Statement of Problem

The goal of this project is to accurately sort 48 pieces, 12 black plastic, 12 white plastic, 12 aluminum and 12 steel, into 4 separate sections of a sorting dish in 60 seconds or less. A maximum of one sorting error is acceptable. The sorting dish is mounted to a stepper motor and thus can be rotated in real time. Items are dropped onto the sorting dish by a conveyor belt, which itself is equipped with an optic and a reflective sensor, both necessary for distinguishing between the 4 piece types. System control is achieved via the ATmega2560 microcontroller, which is programmed in C. A button to pause and resume the project's function and a button to allow the completion of sorting for parts already placed on the apparatus before shutting down function are required. An additional switch is included to act as a kill switch, ceasing all project function instantaneously.

1.2 Design Overview

1.2.1 Conveying Subsystem

The conveying subsystem transports the pieces to be sorted down the conveyor belt through the sensors and is an industrial grade conveyor belt driven by a DC motor. Pulse width modulation is utilized, via the microcontroller, to drive the motor with a change in duty cycle percentage changing the speed of the motor. Multiple duty cycle percentages were tested to determine the optimal speed for the greatest accuracy in sorting. An image of this subsystem can be seen in Figure 1.

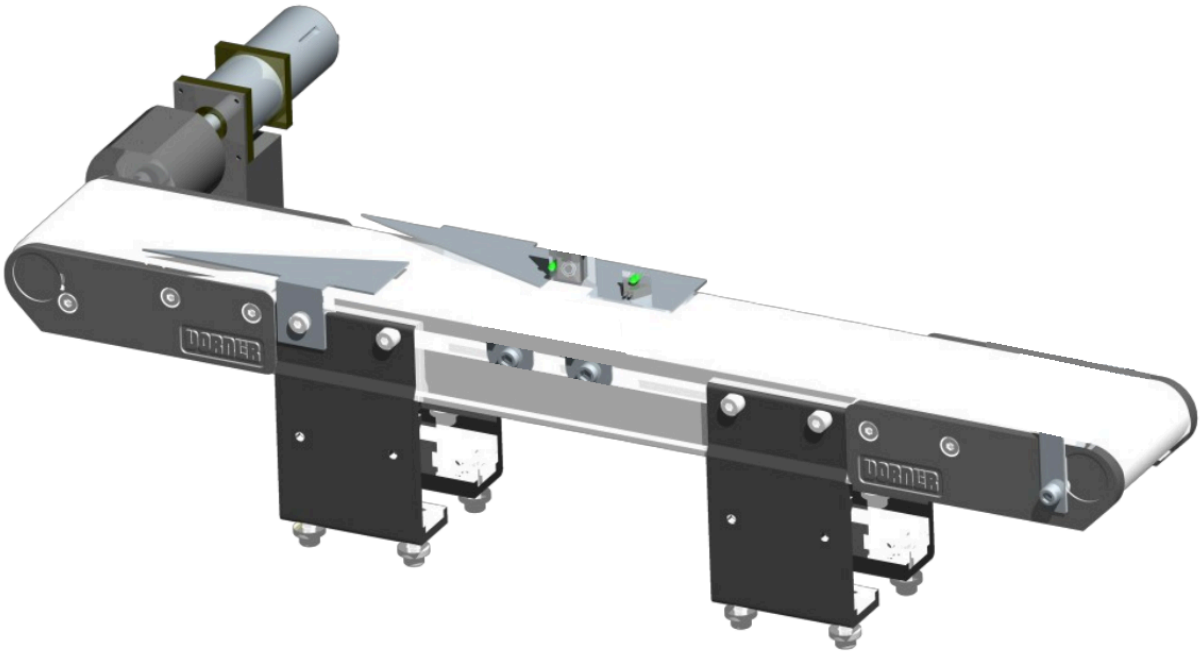

Figure 1.1. Conveying subsystems of the apparatus. Image taken and modified from lecture slides[1].

1.2.2 Inspection Stations

There are two sensors used as part of the inspection station, a reflective sensor and an optical sensor, used to determine the color of each piece. The analog to digital converter of the microcontroller receives values from the sensors with the lowest value read for each part determining the color of the piece. A range of readings for each piece is taken and the reading determined to be at the center, closest to the sensor, of the piece is used to classify the piece. These two sensors and their placement can be seen in Figure 2.

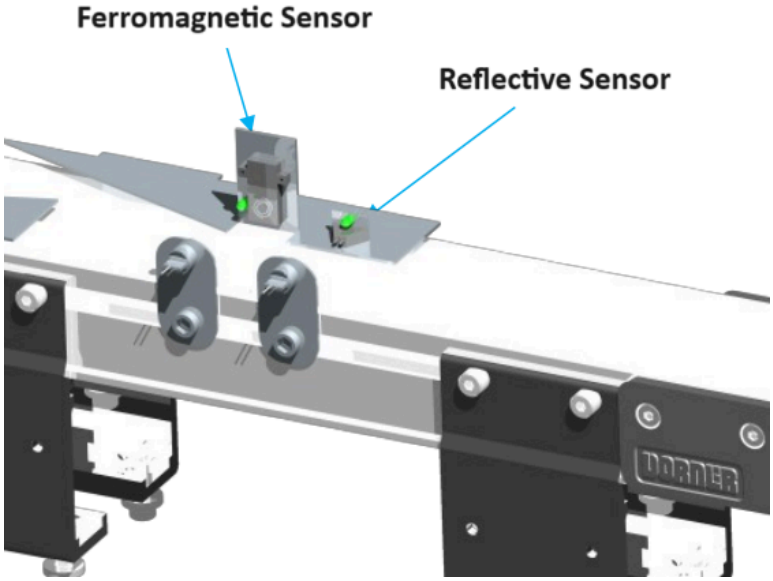

Figure 1.2. Reflective sensor on apparatus. Image taken and modified from lecture slides[1].

The third optical sensor, different from the ones paired with the previous sensor, is contained in an arch-like structure at the end of the conveyor belt which the pieces pass through. This sensor determines when a piece reaches the end of the belt and is ready for placement in the sorting bucket. When this sensor is tripped, the conveyor belt stops and the sorting bucket in the sorting subsystem moves to the correct position for the part to be sorted. The optical sensor can be seen in Figure 3.

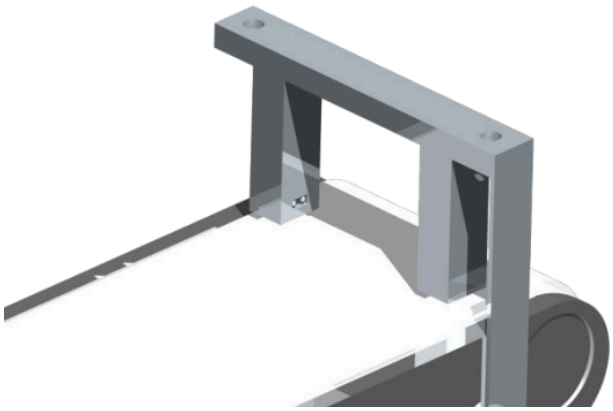

Figure 1.3. Optical sensor at the end of conveyor belt. Image taken from lecture slides[1].

1.2.3 Sorting Subsystem

The sorting subsystem is a round disk structure partitioned into four sections labeled Black, White, Aluminum and Steel. The disk is controlled by a stepper motor which receives signals from the microcontroller to move clockwise or counter clockwise to position the necessary section under the conveyor belt for pieces to be sorted into. There is a Hall effect sensor placed below which uses magnetic flux to trip a switch when both pieces of the sensor are aligned. When the sensor is triggered, the disk stops with the Black section aligned under the conveyor belt. This is used to "home" the disk on Black as a reference point when programming the microcontroller to track disk location during rotation. Figure 4 shows the sorting subsystem with section labels.

Figure 1.4. Top view of sorting disk with labels indicating part placement.

1.2.4 Microcontroller

The microcontroller used is the ATmega2560, an 16 MHz CMOS 8-bit microcontroller. It is programmed using C with Atmel Studio 7.0. The microcontroller has built-in timing registers that are utilized to create multiple timers; one for a delay, one to track the travel time of an item on the conveyor belt, one to track width of a piece inside the reflective sensor and one for debouncing the exit sensor. There are seven external interrupts on the microcontroller, at least five are utilized for the Hall effect sensor, kill switch, pause and resume button, exit sensor, and ramp down feature. The analog to digital converter and the timers utilize an internal interrupt. The analog to digital converter converts analog signals from the reflective and ferromagnetic sensors to an integer value from 0 to 1023 that is used for comparisons. The pulse width modulation feature of the microcontroller is utilized to control the speed of the motor to the conveyor belt.

1.2.5 Output Information

A liquid crystal display(LCD) is used to print various information during the sorting of each piece. Upon program initialization, the LCD will indicate the program is waiting for the sorting disk to home, and then indicate when this process has been completed. Then the LCD indicates the program has entered the sorting process. When the sorting process has completed, the ramp down button has been pressed, or if the pause button has been pressed, the LCD will print the total number of pieces sorted and how many of those pieces are plastic or steel or aluminum. The LCD also indicates when the kill switch has been flipped or if a bad interrupt has been detected. The LCD is run with a 5V voltage supply and input/output pins set for output from the microcontroller.

1.2.6 External Control Features

There are two push buttons and a DIP switch utilized for program control. The DIP switch is used as the kill switch to immediately cease all program functions. The switch is wired active low and is connected to an interrupt of the microcontroller that is set to trigger on either a rising or falling edge. The two push buttons are used for the pause and ramp down features. The pause button is wired active high and connected to an interrupt of the microcontroller set to trigger on a rising edge. The ramp down button is wired active low and connected to an interrupt of the microcontroller set to trigger on a falling edge.

2.0 Methods and Design

2.1 Stepper Motor

The stepper motor is used to move the sorting plate so the corresponding section is below the conveyor belt for a piece to be dropped. The Hall effect sensor below the plate is used to initialize the plate with black centered under the conveyor belt. A while loop is used to to move the stepper and disk until an interrupt is thrown indicating the Hall effect sensor has tripped low. Once this happens, a flag is set to indicate the steppered has “homed” and will break out of the loop stopping the stepper. The interrupt for the Hall effect sensor is then disabled to prevent it from tripping again when sorting pieces.

2.2 Belt Drive Motor

The main benefits of using this motor and driver were design simplicity and cost; that said, the deciding factor was that there were no other options available for the belt drive system.

2.3 Control System

System operation was controlled by a program running on the microcontroller. Code was written in C.

2.3.1 Software Tools

The Atmel Studio 7.0 IDE was used to write and build C code, and AVRDUDE was used to upload programs to the microcontroller. Atmel Studio 7.0 was extremely helpful in the development process as it came packaged with the AVR Libc library, which provided a great multitude of convenient macros and functions. In particular, registers and pins could be accessed by name. This library feature accelerated the development process by creating consistency between the datasheet and the code.

2.3.2 Software Development Methodology

Software for this project was developed using a modular approach. Individual components were tested independently before being integrated with the main program. This approach facilitated debugging and performance validation.

2.3.2.1 Stepper Motor

When sorting a piece, the code keeps track of the current position of the sorting to compare to the position required during identification of the piece waiting to be sorted. This is done to ensure the most efficiency when moving the stepper and disk. If a 180 degree turn is needed, the stepper will continue rotating in the same direction as the previous movement, clockwise or counter clockwise, to avoid delays due to change of direction. The steppers speed is controlled with delays between each sequence step iteration to accelerate and decelerate the stepper, this is explained in detail in section 3.2.1. Additional delays are used with the exit sensor to drop the pieces in the center of the sorting disk before the stepper moves to the next piece's position. The delay is determined by the amount of time required for the belt to turn back on and the piece to travel from the exit gate to the correct sorting disk section below. This is done to ensure accurate placement of the piece and avoids the disk moving to the next position too soon resulting in pieces potentially hitting the section dividers or falling into the wrong section entirely.

2.3.2.2 Belt Drive Motor

Controlling the belt drive motor was straightforward. As mentioned in subsection 2.2, starting and stopping the motor was realized by toggling a single bit in software. PWM was needed to power the motor, and thankfully the microcontroller provided native support for this through any of its 6 timer/counter registers. The process for creating a PWM signal with a counter is shown below in Figure 2.1.

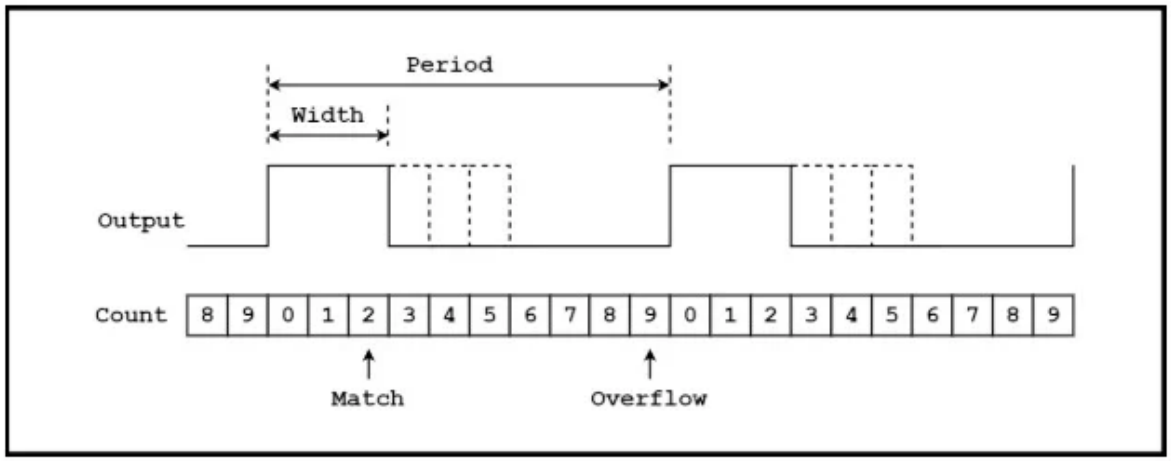

Figure 2.1. PWM signal generated by counter[2].

The "PWM mode" of the microcontroller's counters automated the setup of a counter-based PWM signal, and allowed easy modification of key parameters such as duty cycle and frequency.

2.3.2.3 External Interrupts

External interrupts were necessary for providing a consistent and responsive user interface. The killswitch and pause/resume button in particular necessitated the use of interrupts, for a polling-based architecture would be too unpredictable given the nature of the program. This was mainly due to the stepper motor whose regular rotations all required significant CPU time, thus making simple polling methods unreliable. However, the use of external interrupts presented other challenges, chiefly the disruption of loops (and especially timer-based loops) in the main code. Nonetheless, it was thought the design complexity incurred by this issue would be less than that of a polling-based architecture, and furthermore that the interrupt-based architecture would provide better overall performance and reliability.

2.3.2.4 ADC Conversions/Item Identification

Converting analogue sensor readings to digital numbers was a system-critical function. In particular, the ADC was needed to read the reflective sensor, and therefore to differentiate between item types. Robustness was prioritized, as there were several factors which tended to interfere with this process, namely:

1. the irregular nature of sensor output (when graphed, it looked more akin to a stock market trend line than a smooth parabola);
2. stopping and starting of the belt during item processing; and
3. interrupts firing during item processing, especially those with lengthier service routines.

It was thought that a two-stage processing sequence would be needed to mitigate potential inconsistencies: first, a fixed time period of successive ADC conversions (to handle problem 1); and immediately after, a variable-length extension of this period depending on whether an item

was still being detected. The first stage was designed to protect against the irregularities in sensor output. While polling the sensors was generally effective on its own, occurrences were observed where the polling routine would finish prematurely due to fluctuations in the sensor reading. Employing a timer-controlled period ensured that premature completion would never occur due to variance in sensor readings. The salient shortcoming of this timer-controlled routine was that belt stoppages during processing would cause the timer to complete before the item had fully moved through the sensor. This was remedied by adding the second stage, which ensured which had not fully passed through the sensor would continue to be processed after the timer had completed. While this second stage was theoretically prone to all the same problems as a polling-based architecture, it was found that the first stage always allowed items far enough into the sensor's field of view that variances in the reading would not be great enough to indicate that no item was present. Upon completing both stages only the peak sensor reading was retained, thus providing optimal accuracy in regards to distinguishing between item types.

2.3.2.5 LCD Output

The LCD was used extensively in validating the performance of the ADC. The ability to see what values were being produced during the item processing stage was invaluable to ensuring correct operation. Furthermore, once the typical value range for each item type was known, the LCD provided a means to see in real time how the item had been categorized by the system.

2.3.2.6 Item Tracking

Items were tracked both positionally (relative to the belt) and categorically (relative to each other). Knowing the location and type of an item on the belt was essential to correct operation. Item tracking began at the first optical sensor, which was positionally across from the reflective sensor. Whenever this sensor was tripped, it initiated the two-stage item processing routine. Upon completion, the type of the item became available and needed to be stored. Storage was accomplished via linked list. In particular, the list was set up to operate as a FIFO queue, since items would always need to be removed in FIFO order. Every time a new item was processed, it would be stored at the back of the queue. Upon nearing the end of the belt, items would trip the exit sensor (a second optical sensor), thus initiating the final phase of the sorting procedure. An interrupt would be fired, and the exit sequence would begin:

1. stop the belt;
2. retrieve the item type by popping the queue;
3. if necessary, rotate the stepper;
4. disable this interrupt and clear pending flag;
5. start a timer and enable its interrupt; and
6. resume the belt.

Upon resuming the belt, positional tracking for the recently sorted item would cease and the item would be recorded in the storage log. The timer was implemented as a means to compensate for

the electromagnetic noise produced by the belt drive motor. This noise frequently caused the exit timer to trip twice for a single item, which was a serious problem. The timer effectively enforced a minimum duration between exit sensor interrupt events, which in turn mitigated mistriggers at the exit sensor. When the timer finished its count, it would trigger an interrupt whose service routine re-enabled the exit sensor interrupt and disabled the timer interrupt.

2.3.2.7 System Calibration

Accurate item identification required the system to be calibrated before use. External factors such as ambient lighting and item-to-item differences resulted in variances in the reference parameters of the tracking system. Three types of calibration were necessary before operating the system:

1. no-item calibration;
2. conversion timer calibration; and
3. item calibration.

No-item calibration simply meant observing the reflective sensor output with no items present. It is important to mention here that the reflective sensor outputs lower values for higher reflectivity. The purpose of no-item calibration was to determine the lowest value that could be produced when no item was present, and also to estimate maximum deviance of this value over a short timespan (~30 minutes).

Timer calibration was designed to determine the smallest value for the ADC timer such that the peak sensor reading would always be captured for an item passing through continuously (i.e. without the stopping).

Item calibration was designed to determine the range of values each item type could take on, therefore allowing items to be correctly identified.

These 3 calibrations were designed to be carried out sequentially. Timer calibration relies on values obtained from no-item calibration, and item calibration relies on values obtained from timer calibration. Altogether, system calibration was intended to ensure items would be correctly identified as they traversed the conveyor belt.

2.3.2.9 Sorting Algorithm/Main Loop

With all the above operating in unison, the system is capable of sorting. An algorithm in the main loop of the program coordinates the different tasks that must be carried out by the system and ensures that all actions are carried out at the appropriate time. This algorithm is detailed in section 3, but a simplified summary is as follows:

- when an item trips the first optical sensor:
    - begin item identification routine

- when an item trips the exit sensor:
    - stop the belt
    - remove the first item in the queue
    - rotate the stepper
    - resume the belt

Upon tripping, sensors cause interrupts to be fired which update flags representing various system states. Polling these flags in the main loop allows the system to respond to events.

3.0 Results

3.1 Technical Descriptions

3.1.1 Stepper Motor

The stepper motor is a Soyo 6V 0.8A Unipolar Stepper Motor[4] with a step angle of 1.8 degrees. The stepper has an angle accuracy of ±5% when the motor is unloaded and running in full step mode. For the purposes of this design, the stepper will be run in full step mode and have some loading due to the sorting disk and the pieces that have been loaded onto the disk. The stepper is driven by a L298 H-Bridge Motor Driver[4]. This driver is equipped with LEDs indicating which stepper coils are active during step sequencing. This feature is used to troubleshoot and ensure the motor is running in the correct mode. The motor driver also has a built-in thermal-shutdown feature so if the stepper becomes overloaded it will slow down and stop instead of overheating.

3.1.2 DC Motor

The conveyor belt was driven by a 5V brushed DC motor, which was controlled via the Pololu VNH2SP30 motor driver. The Pololu allowed the motor to be powered via PWM (up to 20kHz), and to be controlled with a 4-bit code:

INA	INB	ENA	ENB	Function
1	1	1	1	Stop
0	1	1	1	Forward
1	0	1	1	Backward

Table 3.1 Belt drive motor control codes.

3.1.3 Sensors

The Hall effects sensor used with the stepper motor is a solid state SS400 series sensor from Microswitch. This is a magnetic flux sensor that is powered by a 6.5V source and is active

low. The sensor is active low and connected to an interrupt of the microcontroller that triggers on a falling edge.

The reflective sensor is an OPB74XW, X being a model number between 0 and 4 because the exact model number used is unknown, Reflective Object Sensor. It produces a voltage between 0V and 5V when an object is in front of it, with 0V indicating most reflective and 5V least reflective. The sensor is powered with an external, to the microcontroller, 4V battery supply. The voltage from the sensor is read by the ADC and converted into a 10 bit value.

There are two optic sensors used. The exit sensor is an OPB819Z Slotted Optical Switch powered by a 5V supply external to the microcontroller. The exit sensor is an active low sensor and triggers a microcontroller interrupt on a falling edge. The optic sensor paired with the reflective sensor is a Silicon NPN Phototransistor that is active high and triggers a microcontroller interrupt on a rising edge.

3.1.4 Technical Diagrams

Figures 3.1 and 3.2 below, are visual representations of the system as a whole and detailed representation of the circuitry used.

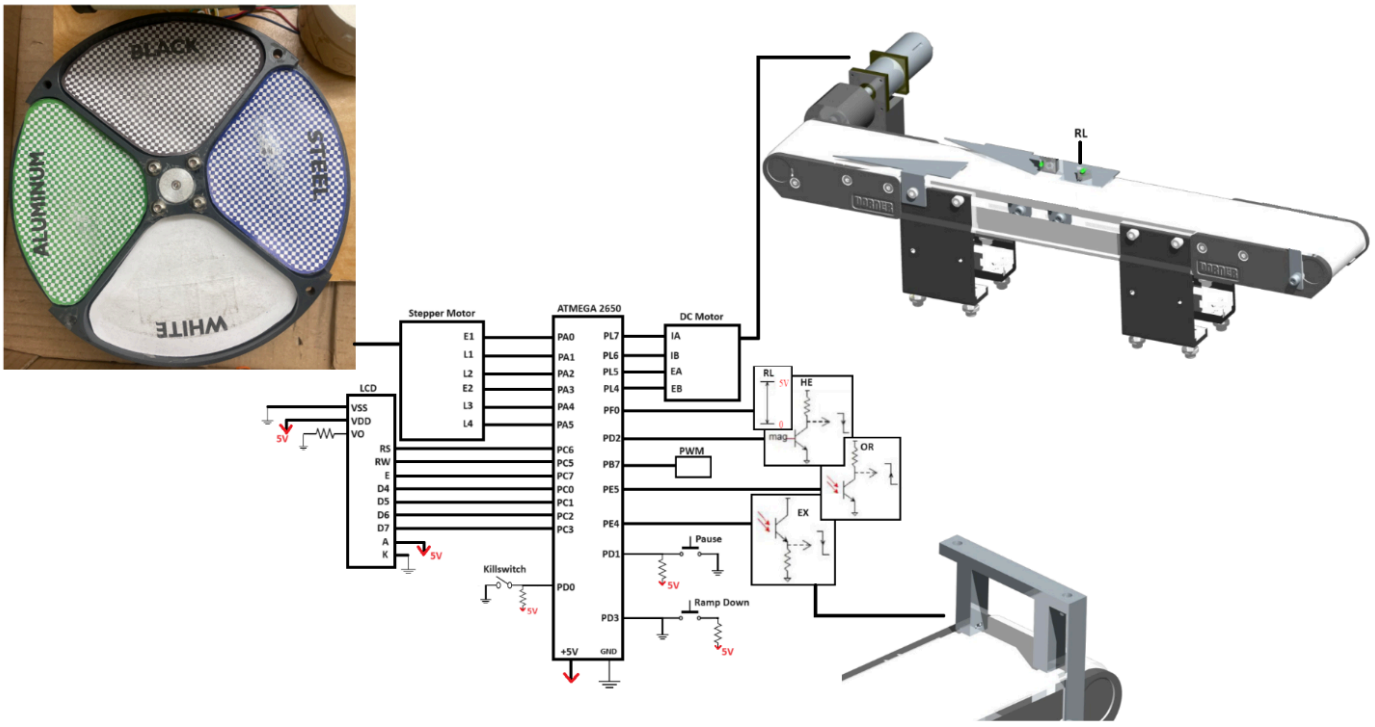

Figure 3.1. High level system diagram.

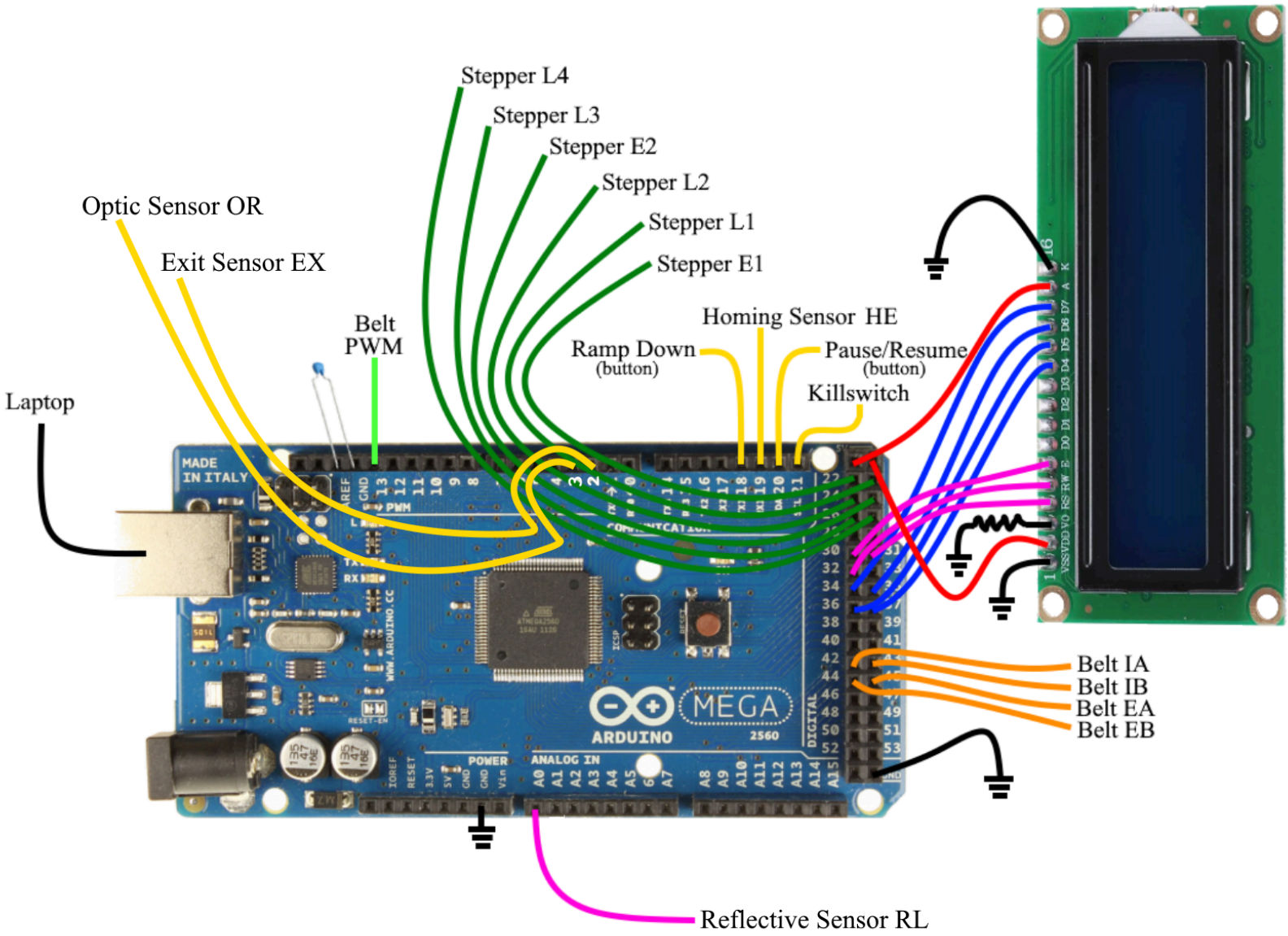

Figure 3.2. System circuit diagram.

3.2 System Performance Specifications

3.2.1 Stepper Motor

The stepper motor is designed to operate in 2 Phase Dual Coil Mode, which enables both coils at the same time. Enabling both coils increases the torque initiated in each step, compared to single phase mode, allowing for smoother and faster stepping of the motor. The sequencing table for the stepper motor can be seen in Table 3.2.

Step	EN1	L1	L2	EN2	L3	L4

1	1	1	0	1	1	0
2	1	0	1	1	1	0
3	1	0	1	1	0	1
4	1	1	0	1	0	1

Table 3.2 Stepper motor sequencing in 2 phase dual coil mode.

Delays were used between each step to allow time for the sorting disk to physically move. Longer delays were used at the beginning and end of a motion sequence to allow the stepper to ramp up in acceleration to avoid the physical skipping of steps. An array was used to achieve this with the delay values decreasing as the stepper iterates through a complete motion until reaching a minimum delay value that does not induce step skipping. The minimum value is maintained until the final few steps of a motion where the end of the array will mirror the start of the array. Sample code from this method can be seen in Figure 3.3.

```
int delay_a[50] = {15,15,13,12.5,12,10,9,8,7,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6
                  ,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,6,7,8,9,10,11,12,12.5,13,15,15};
```

Figure 3.3. Sample code for delay array from Stepper.h file, used to ramp up and down stepper speed for a 90° turn.

The method of controlling the stepper speed was chosen based on the Trapezoidal Acceleration model seen in Figure 3.4. Where the section between T1 and T2 represents the maximum speed of the motor, smaller delay values, and starts from a speed of zero and increases to T1 and decreases from T2 back to zero at T3. As the speed increases the delay value times decrease and and the speed decreases the delay value times increase to the initial delay value.

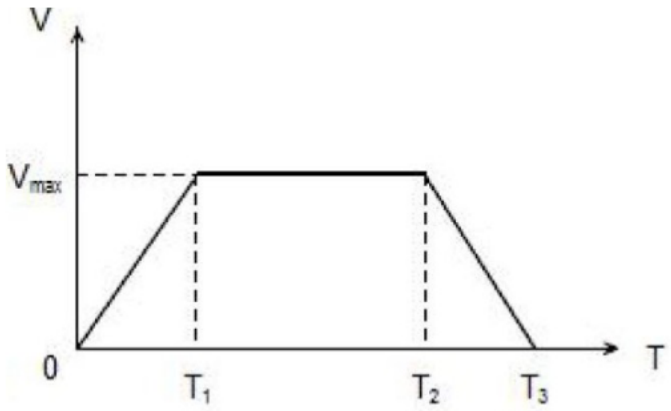

Figure 3.4. Trapezoidal acceleration model for a unipolar stepper motor.

3.2.2 Belt Drive Motor

Unfortunately, the construction of a brushed DC motor presents inherent challenges to an electrically sensitive system. The commutator brushes on these motors produce electrical arcing, which has the effect of generating high frequency electromagnetic noise. This phenomenon had a

significant effect on the sensors of the system, which in turn added complexity to the design of the system. Solutions to that problem are discussed later in this report. The motor was powered with a 38% duty cycle 3.9kHz PWM signal, generated by the microcontroller. While the motor was rated to support up to 20kHz PWM, code for a 3.9kHz PWM signal was reused from a past experiment, and testing results indicated that this frequency provided excellent performance. It was found that a 38% duty cycle provided the optimal speed in regards to minimizing errors while maintaining performance. The system was tested with duty cycles of up to 49%, but values larger than 38% were found to be error-prone. At these speeds, pieces could fall off of the belt in an unpredictable and uncontrolled manner, particularly steel pieces as they carried the most momentum. Furthermore, abrupt stopping and starting of the belt could cause pieces to be jostled, which could skew readings from the reflective sensor. These issues were compounded by the stepper motor which struggled to keep up with piece throughput at higher belt speeds, thus incurring another degree of unpredictability for the system.

3.3 Sorting Algorithm

Figure 3.5 below shows a flowchart describing the algorithm deployed in the main loop of the sorting system.

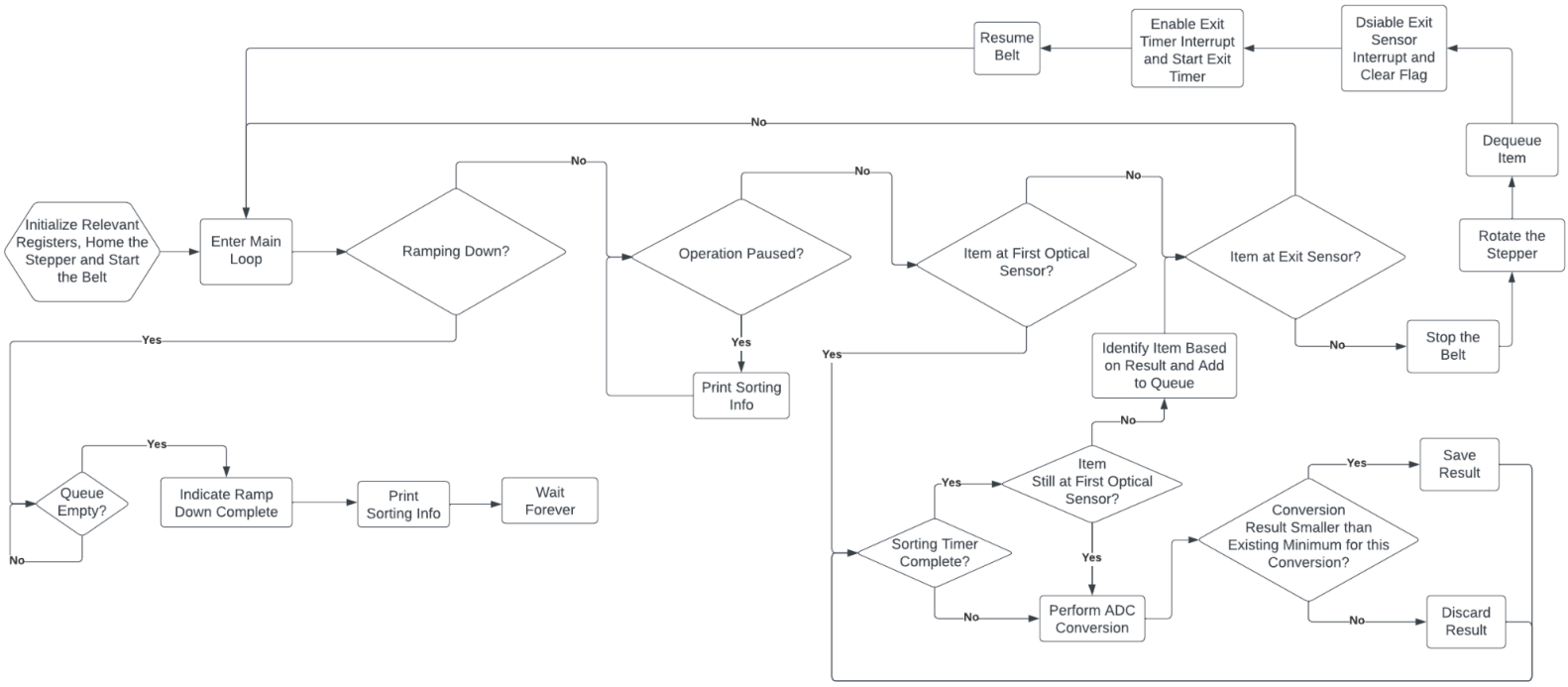

Figure 3.5 Sorting algorithm flowchart.

3.3.1 Interrupts

Several interrupts, both internal and external, were used in order to convey status updates to the main loop. The following subsections detail the service routines of each of these interrupts, in order left to right relative to the flowchart in Figure 3.3.

3.3.1.1 Stepper Homing Interrupt

The stepper must be homed during the initialization phase. This is accomplished by using a while loop to step through the stepper sequencing table, Table 3.2, and a flag to indicate completion of the homing function. When the Hall effects sensor becomes active, interrupt `INT2` is triggered and the homing flag is changed to TRUE. The interrupt is then deactivated to avoid future triggering, and the while loop stepping through the stepper sequence is exited with no further steps executed.

3.3.1.2 Ramp Down Interrupts

The ramp down interrupt, `INT3`, is triggered by the press of a button. Upon triggering for the first time, the service routine updates the status flag `ramp_down` to indicate that ramp down procedures have been initiated. This status update is reflected on the LCD, where it is displayed that the system has entered ramp down mode. The service routine then starts an 8.4s timer, `TIM5`, and enables an interrupt to fire upon completion of this timer. Subsequent presses of the ramp-down interrupt have no effect. Nothing changes in the main loop yet. When the ramp down timer completes and its interrupt fires, the corresponding service routine disables the ramp-down timer interrupt and updates the second ramp down status flag, `finishing`. This flag is visible to the main loop and causes the ramp down procedure shown in Figure 3.3.

3.3.1.3 Pause/Resume Interrupt

The pause/resume interrupt, `INT1`, is triggered by the press of a button. Upon triggering, the service routine checks the `running` status flag to determine whether the system is currently running. If yes, the routine pauses the belt; if no, the routine resumes the belt. In either case, the `running` flag gets updated. This flag is visible to the main loop and is responsible for pause behavior therein.

3.3.1.4 First Optical Sensor Interrupt

The first optical sensor interrupt, `INT5`, is triggered by a rising edge on the output of the first optical sensor. Upon triggering, the service routine sets the `inbound` status flag, which is visible to the main loop and indicates that an item has passed the first optical sensor. The `inbound` flag is cleared upon successfully processing an item within the main loop.

3.3.1.5 ADC Conversion Complete Interrupt

The ADC conversion complete interrupt is triggered by the completion of an ADC conversion. Upon triggering, the service routine stores the conversion result within the `ADC_result` variable and sets the `ADC_result_flag` status flag. This status flag is visible to the main loop: while performing conversions, the loop waits for the `ADC_result_flag` flag to be set, then evaluates the value and clears the flag.

3.3.1.6 Exit Interrupts

The exit sensor interrupt, `INT4`, is triggered by a falling edge on the output of the exit sensor. Upon triggering, the service routine sets the `existing` status flag, which is visible to the main loop and indicates that an item has tripped the exit sensor. The main loop clears this flag after successfully handling the exit process. The exit timer interrupt, `TIM4`, is enabled in the main loop during the exit process. Upon triggering, the interrupt disables itself and re-enables the exit sensor interrupt.

3.3.1.8 Killswitch Interrupt

The killswitch interrupt, `INT0`, is triggered by a rising or falling edge. Upon triggering, the service routine stops the belt, prints "Kill Switch Hit" to the LCD and then enters an infinite loop, thus stalling the program until the microcontroller is reset.

3.3.2 Notes on the Item Identification Process

This section presents certain noteworthy details regarding the workings of the item identification process.

3.3.2.1 Regarding the ADC Conversion Timer

While most of the timers used in the sorting algorithm rely on interrupts to indicate their completion, the ADC timer does not. Instead, the `OCF3A` bit, which indicates the completion status of the `TIM3`, is polled. This bit is updated automatically by hardware upon timer completion, and therefore no interrupt is necessary here.

The duration of the ADC conversion timer is variable, and its value depends on calibration values. However, the time duration of this timer can be evaluated by dividing the value defined by the `ADC_STOPWATCH` macro by 125,000. This is because the timer counts the ADC_STOPWATCH number of ticks at a frequency of 125kHz.

3.3.2.2 Regarding the First Optical Sensor

While there is an interrupt associated with the first optical sensor, this sensor also gets polled during item identification. Once the ADC conversion timer completes and stage two of

item identification is entered in the main loop, ADC conversions continue to run as long as the output of the first optical sensor reads a logical high.

3.4 Calibration

Proper calibration underpinned the functionality of the sorting system. Moreover, ensuring that accurate reflectivity-sensor values were being obtained was essential to the identification process, and therefore also to correctly sorting items. Several measures, detailed in the following subsections were taken to validate that accurate values were indeed being obtained.

3.4.1 No-Item Calibration and ADC Conversion Timer Calibration

The no-item calibration exists solely as an intermediate step in the ADC conversion timer calibration. ADC conversions are run continuously during no-item calibration (taking values from the reflectivity sensor as input), but only the lowest conversion value is shown on the LCD. The intention of performing this calibration is to find the smallest sensor value that can be produced when no items are present; or, in other words, to find the peak ambient reflectivity value. This calibration produces another value as a byproduct: the deviance one can expect in ambient reflectivity values. To illustrate this, consider performing this calibration once at t+0 minutes and again at t+30 minutes. Suppose there is a difference of 7 between these two values (this was found to be typical): then, one can expect that the ambient peak may fluctuate by +/-7 every 30 minutes. Of course, this value is approximate, but can still give some idea of how these values may change over time.

How does this relate to ADC conversion timer calibration? The working principle of the ADC conversion timer calibration process is this: if the reflectivity sensor value is equal to or greater than the value given by [(ambient peak) - (ambient deviance)] for a long enough period of time while the belt is running, it can be concluded that an item that was passing through the reflectivity sensor has fully exited the domain of the sensor. As stated earlier, the output of the reflectivity sensor is not a clean parabola, but rather a jagged curve that oscillates up and down as it trends up or down. Given the extreme frequency of the CPU clock compared to the frequency of these oscillations, it is possible for an item to produce sensor values indicating there absence of an item for a relatively long time. At 38% belt speed, it was observed that items could produce such values for up to 4,000 consecutive ADC conversions. The goal of performing the ADC conversion timer calibration is to determine how long it takes for an item to pass by the reflective sensor at any given belt speed. To do this, there must be a clear definition of what constitutes an item having passed through. It was found that, for a continuously running ADC, a sequence of 5,000 ADC conversion values all equal to or greater than the value given by [(ambient peak) - (ambient deviance)] was a sure indicator that there was no item in front of the sensor. With this in mind, the timer calibration algorithm proceeds as follows:

1. Wait for the first optical to be tripped.

2. Start 125kHz timer.
3. Perform ADC conversion:
    a. if the result is equal to or greater than [(ambient peak) - (ambient deviance)], increment a designated counter variable.
    b. if the result is less than [(ambient peak) - (ambient deviance)], reset the designated counter variable to zero.
4. If the value of the designated counter variable is less than 5,000, return to Step 3.
5. Store the value from the 125kHz timer.
6. Repeat 9 more times from Step 1.
7. Output [(smallest timer value) - (5,000)]/2 to the LCD.

The calculation in Step 7 computes the recommended value for the ADC conversion timer. It provides a good approximation for how long it takes for an item to pass the halfway point of the reflectivity sensor; or, in other words, how long it takes for the system to extract the peak sensor value for that item. The output value can be divided by 125 to get the actual time in ms. One should note that the difference computed in Step 7 is not a difference between quantities of the same type. The (smallest timer value) is a quantity representing milliseconds scaled by a factor of 125, and 5,000 represents an unknown (but thought to be between 20 and 30) number of CPU clock cycles multiplied by a factor of 5,000. Nonetheless, rigorous testing proved that the value produced by this equation is solid and reliable for the purposes of calibrating the ADC timer. Once this value has been computed, the user should set the `ADC_STOPWATCH` macro accordingly. This macro determines the duration for which the ADC will perform continuous conversions in the first stage of the item identification process. To summarize, timer calibration involves running 10 items through the first set of sensors (the reflectivity sensor and the optical sensor), and using the data gathered from this to determine the maximum elapsed time between an item tripping the first optical sensor and the system extracting the peak reflectivity-sensor value for that item.

The attentive reader may notice that the value computed during ADC timer calibration is only meaningful in the case where the belt is running continuously (i.e. with no stoppages); and this is indeed true. This is the reason why the item identification process has two stages.

3.4.2 Item Calibration

Item calibration must be carried out after completing the ADC timer calibration. The purpose of item calibration is to determine the range of sensor values each type of item can be expected to produce. Since the final goal of this project was to sort 12 pieces of each type, item calibration was designed to work with 12 items at a time. The item calibration algorithm is as follows:

1. Wait for an item to trip the first optical sensor.
2. Start the ADC conversion timer.

3. Perform continuous ADC conversion until the timer is complete, storing the conversion result only if it is smaller than the current smallest value.
4. Store the resultant value.
5. Repeat 11 more times from Step 1.
6. Print the smallest and largest values to the LCD.

The item calibration algorithm effectively preserves the peak sensor readings for each of the 12 items, then displays the range of these values. Once this calibration is repeated for all four item types, the user can know what the smallest and largest expected values for each item type are, and set the `THRESHOLD` macros appropriately. These macros are ultimately used to identify items in the main loop.

3.5 Testing, Limitations and Tradeoffs

Testing the functionality of the system was carried out simply by running items through over and over again, and identifying points of failure. Investigations of the underlying cause were carried out when such points arose, and resolutions were made as necessary. Different item types were found to pose different problems. For example, steel items carried much more momentum than plastic or aluminum, and thus were more greatly affected by abrupt stops at higher belt speeds. When running at 49% belt speed, steel blocks would often get thrown off the end of the belt if they weren't spaced far enough apart. Black plastic and white plastic items would regularly get misidentified by the system since their values were so close together. Aluminum items, on the other hand, never caused any issues: their light weight and easily distinguishable reflectivity value played nicely with all parts of the sorting system. Eventually, testing results conclusively indicated that the system could consistently identify all item types and drop them into the appropriate bin. At this point, testing became focused around different permutations of items, and item throughput. It was found that permutations which caused the stepper to reverse its direction of travel (for example, a clockwise turn followed by a counterclockwise turn) could cause the stepper to lock up and fail. This was resolved by inserting small delays which allowed the stepper to recuperate for a moment before launching into the next turn. While this solved the initial problem of locking up, it introduced inefficiency into the system as such a delay was not necessary for most turns. The code was augmented such that the function responsible for rotating the stepper would return an integer which encoded the type of turn that had to be performed: no turn, a quarter turn, a half turn, or a direction-reversal turn. All of these cases, even no turn, required some delay in order to achieve flawless performance, and these parameters were tweaked and adjusted until the optimal settings were found. Increasing item throughput posed even more challenges. Item identification and exit processes were found to interfere with each other rather heavily, and this was very problematic. Whereas other issues had arisen and subsequently been resolved, no satisfactory resolution could be found here. Instead, it was discovered that the problem could be avoided as long as items were spaced at least 2 inches apart, and this was what was done during the final demonstration.

Once essential system functionalities (that is, correctly sorting an arbitrary number of items in any given permutation) had been thoroughly validated, attempts were made to optimize the performance of the system. The system was initially designed with reliability in mind, not performance: it was thought that once the system had been proven its reliability under all circumstances, improving performance would be as simple as adjusting parameters. This was, unfortunately, a false pretense. Nearly all parameter adjustments instigated problems elsewhere in the system, and ultimately the only optimizations that could be made without jeopardizing the system's reliability were in the speed of the stepper motor.

The root of the problem ended a sort of paradoxical interaction between the item identification process and the exit process. The exit process, responsible for rotating the stepper motor, needed to be able to deploy delays *while the belt was running* in order to support good item throughput. However, since the item identification process scanned items using a polling-based architecture, allowing the exit sensor to deploy delays while the belt was running meant allowing the exit sensor to move half-scanned items out of the reflectivity sensor before item identification was complete. While the inability for the exit process to deploy delays with the belt running incurred significant performance penalties, allowing it to do so effectively broke an integral part of the sorting system; that is, the item identification process. The reason it was so important for the exit sensor to be able to deploy delays while the belt was running was because it was the only mechanism for exerting direct control over how items rolled off the belt. Without these delays, when two items were too closely spaced, the exit sensor would trip for the second item before the first item had finished rolling off the belt, thus resulting in both items being dropped in the bin meant only for the second item. As mentioned above, this could be remedied by having the items spaced a couple inches apart; but, it came at a great cost in performance. Item spacing was not the only performance cost incurred by this conflicting design: faster belt speeds, which were initially intended to be used for a performance boost after the system had proved its reliability, were no longer an option because the items would have to be spaced even further apart to compensate for the same issue. Moreover, the design of the code inherently enforced a minimum time interval between items on the belt. This throttled the throughput of the system, resulting in a reliable but poor-performing system.

Though none succeeded, attempts were made to rewrite the program in such a way that delays could be deployed by the exit process without interfering with the item identification process. In particular, mechanisms for completing a half-finished item identification from within the exit process were prototyped, but none accomplished the desired effect. Several different methods were tried in the effort to tackle this problem including copying the second stage of the item identification process into the exit process, and forcing the exit process to use a timer-interrupt based delay as opposed to an mTimer function call, thus forcing it to relinquish its control immediately after starting the timer as opposed to waiting to do so until the timer had completed. In spite of all efforts, no solution could be found that did not break the system in

some other way, and eventually it was conceded that this problem could not be solved within the remaining timeframe of the project.

3.6 Operating Parameters

As stated in section 3.5, few optimizations were successfully implemented. Due to the inherent time requirement between items on the belt, increasing the belt speed yielded no performance benefit and served only to increase the error rate. There was no reason to run the stepper under its maximum safe speed, as this speed was found to be consistently error free. As such, the only performance optimization that could reasonably be made was to decrease the space between the items on the belt. It was found that a space of approximately two inches produced reliable and consistent results (52 seconds and 0 errors). Reducing this spacing to approximately one inch yielded a better time, but with more errors (45 seconds and 3 errors). Overall, it is not recommended to feed items at a spacing any smaller than two inches. The marginal performance improvement achievable by doing so was not deemed to be worth the degree of inconsistency it introduced.

The main adjustable system parameters are defined by macros at the top of the program. The parameters which yielded no errors, and thus the recommended parameters, are as follows:

- `NO_ITEM_THRESHOLD` 961
- `ALUMINIUM_MAX` 150
- `STEEL_MAX` 700
- `WHITE_MAX` 897
- `BELT_SPEED` 38
- `ADC_STOPWATCH` 6483
- `NO_TURN_DELAY` 20
- `QUARTER_TURN_DELAY` 10
- `HALF_TURN_DELAY` 100
- `REVERSAL_DELAY` 220
- `EXIT_INT_DELAY` 4000
- `NO_ITEM_TIME` 5000
- `AMBIENT_DEVIANCE` 7

Some of these macro names are inconsistent with this report. This was done for clarity earlier in the report, but one should note that `NO_ITEM_THRESHOLD` corresponds to the ambient peak, and `NO_ITEM_TIME` is the value used during timer calibration.

3.7 Novel Additions(if any)

It was thought that the two-stage item identification process was a clever way to implement a robust item identification system. Both stages complemented the flaws of the other: the first stage was immune to errors stemming from variances in sensor readings, but unable to

handle start/stop events; the second stage was prone to errors caused by sensor variance, but handled start/stop events without an issue. Not only was this method of item identification effective, it exhibited an interesting duality.

The exit interrupt timer was thought to be another novel addition. While many others solved the problem of the exit sensor tripping twice for one item using hardware, the exit timer used in this project solves the problem in software at the small cost of a few clock cycles. Enforcing a minimum time duration between exit interrupt firing events was a simple way to solve a troublesome issue.

3.8 Experience and Recommendations

Without a doubt the single most significant source of problems during the development of this project was lack of foresight in regards to the interaction of various system components. The interaction between the exit process and the item identification process crippled performance, and this likely could have been avoided if more thought had been put into how the two processes would interact when both tried to run at the same time. This project provided much insight not only into the workings and intricacies of the various system components, but also in regards to how to better carry out the development process in general. Better planning would have greatly benefited the final outcome of this project, and this will be taken into consideration in all future projects.

The ATmega2560 microcontroller and its datasheet, while complex and bizarre on a surface level, proved to be an effective and relatively straightforward tool. Learning how to glean information from the datasheet may have been the biggest learning curve, but once this hurdle was overcome, it became much easier to figure out how to make the microcontroller do what was desired of it.

One final recommendation for any who endeavor to carry out such a project would be to acknowledge the exponential increase in design complexity that comes with operating different subsystems in unison. Over the course of development, many errors arose seemingly out of nowhere upon trying to bring two parts of the system together. Difficulties spawning from this have strong potential to hinder the development process, and can render initial project goals impossible due to the time constraints imposed on the developer.

4.0 Conclusions

This project proved to be quite challenging. While both group members had previous experience designing mechatronics systems, the system designed for this project was significantly more complex than anything previously encountered. In spite of the difficulties and

overall poor performance of the system, the group succeeded in developing a system that fully met the specifications laid out in the design requirements.

5.0 References

[1] Dr. Yang Shi, MECH 458 Lecture 11 Slides,
https://bright.uvic.ca/d2l/le/content/319613/viewContent/2507316/View
[2] Stephen Colley, "Pulse-width Modulation (PWM) Timers in Microcontrollers"
https://www.allaboutcircuits.com/technical-articles/introduction-to-microcontroller-timers-pwm-timers/
[3] Robotshop Inc., “Soyo 6V 0.8A 36oz-in Unipolar Stepper Motor Specifications,”
[4] Solarbotics, “The Compact L298 Motor Driver,” L298 Datasheet, [Revised June 2007].

Appendix A: Original Statement of Preliminary System Design

1.0 Introduction

The goal of this project is to accurately sort black plastic, white plastic, aluminum and steel pieces into 4 separate sections of a dish as quickly as possible. The dish is mounted to a stepper motor and thus can be rotated in real time. Items are dropped onto the dish by a conveyor belt, which itself is equipped with a ferromagnetic sensor and a reflective sensor, both of which are necessary for distinguishing between the 4 item types. System control is achieved via the ATmega2560 microcontroller, which is to be programmed in C. Successful completion of the project requires sorting completion of 48 items in at most 60 seconds with a maximum of 1 sorting error allowed. A button to pause and resume the project's function and a button to allow the completion of sorting for parts already placed on the apparatus before shutting down function are also required. An additional button or switch is included to act as a kill switch, ceasing all project function instantaneously.

1.1 Apparatus

1.1.1 Conveying Subsystem

The conveying subsystem is an industrial grade conveyor belt driven by a DC motor. Pulse width modulation is utilized, via the microcontroller, to drive the motor. The default speed is 50% of the pulse width modulation duty cycle, a change in duty cycle percentage will change the speed of the motor. An image of this subsystem can be seen in Figure 1.

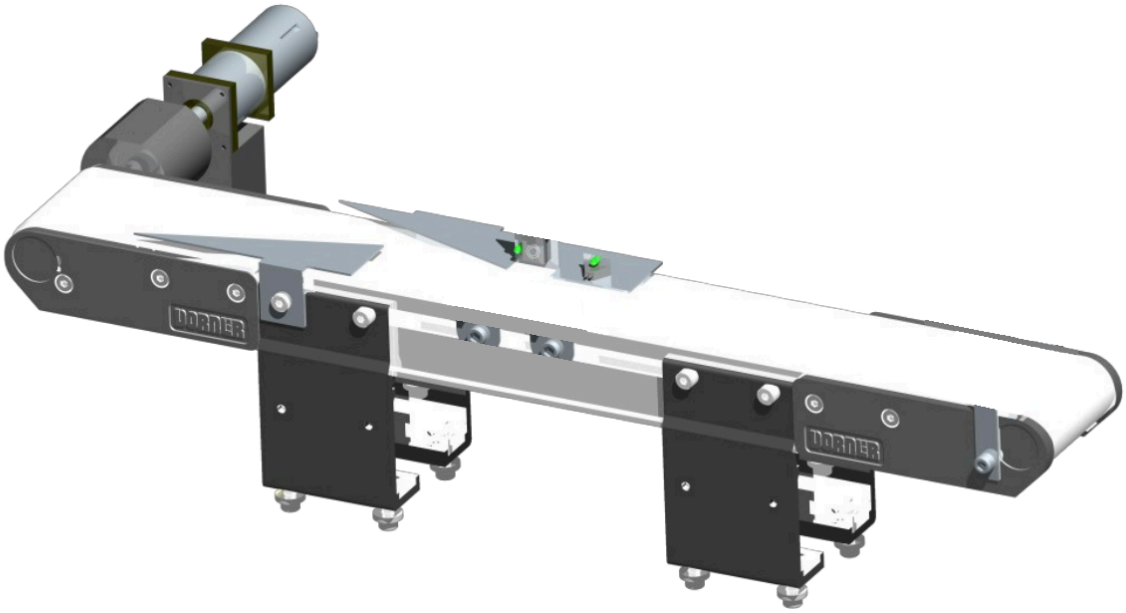

Figure 1. Conveying subsystems of the apparatus. Image taken and modified from lecture slides[1].

1.1.2 Inspection Stations

There are 3 sensors used as inspection stations, a ferromagnetic sensor, a reflective sensor and an optical sensor. The ferromagnetic sensor, paired with an optical sensor, is used to determine the type of material the piece passing through the sensor is. The reflective sensor, also paired with an optic sensor, is used to determine the color of the piece. The analog to digital converter of the microcontroller produces values from the sensors that are used to determine the material and color of the pieces. A range of readings for each piece is taken and the reading determined to be at the center, closest to the sensor, of the piece is used to classify the piece. These two sensors and their placement can be seen in Figure 2.

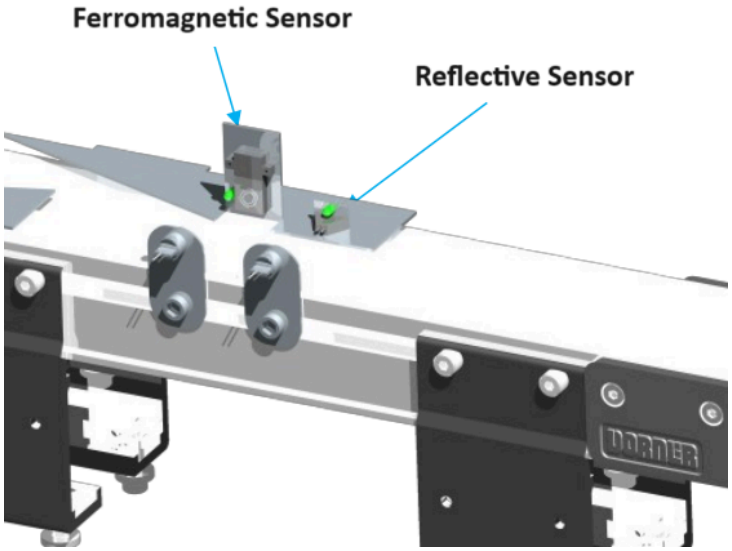

Figure 2. Ferromagnetic and Reflective sensors on apparatus. Image taken and modified from lecture slides[1].

The third optical sensor, different from the ones paired with the previous sensors, is contained in an arch-like structure at the end of the conveyor belt which the parts will pass through. This sensor is used to determine when a part reaches the end of the belt and is ready for placement in the sorting bucket. When this sensor is tripped, the conveyor belt will be stopped so the sorting bucket in the sorting subsystem can be moved to the correct position for the part to be sorted into. The optical sensor can be seen in Figure 3.

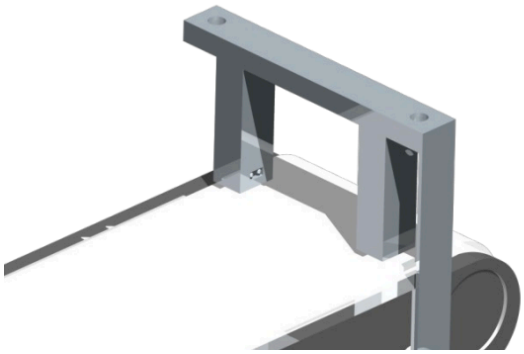

Figure 3. Optical sensor at the end of conveyor belt. Image taken from lecture slides[1].

1.1.3 Sorting Subsystem

The sorting subsystem is a round disk structure partitioned into four sections labeled Black, White, Aluminum and Steel. The disk is controlled by a stepper motor which receives signals from the microcontroller to move clockwise or counter clockwise to position the necessary section under the conveyor belt for pieces to be sorted into. There is a Hall effect sensor placed below which uses magnetic flux to trip a switch when both pieces of the sensor are aligned. When the sensor is tripped, the disk stops with the Black section aligned under the conveyor belt. This is used to "home" the disk on Black as a reference point when programming the microcontroller to track disk location during rotation. Figure 4 shows the sorting subsystem with section labels.

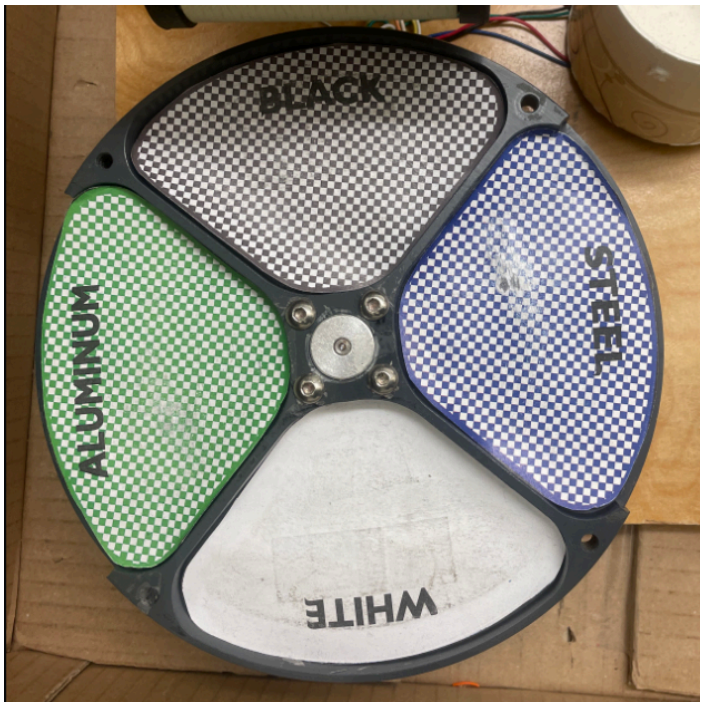

Figure 4. Top view of sorting bucket with labels indicating part placement.

1.2 Microcontroller

The microcontroller being used is the ATmega2560, an 16 MHz CMOS 8-bit microcontroller. It is programmed using C with Atmel Studio. The microcontroller has built-in timing registers that are utilized to create two timers; one for a delay and one to track the travel time of an item on the conveyor belt. There are seven external interrupts on the microcontroller, at least five are utilized for the Hall effect sensor, kill switch, pause and resume button, exit sensor, and ramp down feature. The analog to digital converter and the timers utilize an internal interrupt. The analog to digital converter converts analog signals from the reflective and ferromagnetic sensors to an integer value from 0 to 1023 that is used for comparisons. The pulse width modulation feature of the microcontroller is utilized to control the speed of the motor to the conveyor belt.

1.3 Potential Challenges and Solutions

Some of the potential challenges we anticipate facing are fluctuating values from the sensors and the ranges for the material type and color overlapping, the stepper motor skipping steps when speed is increased, issues with the interrupts and determining priority, and consistency of access to the same apparatus for testing. We will attempt to circumvent these issues by calibrating with as many ideal pieces as possible, slowing the motor speed when starting and changing directions, implementing code to track when and which interrupts are firing during testing, and getting as much testing as possible done on the apparatus while it is available and writing test code utilizing the LED's to confirm expected function before testing on the apparatus.

2.0 Timeline

Figure 5 is a chart of the estimated timelines for each major task and goals for the dates to start each task. Tasks assigned a risk value, are tasks that have not been started yet and the risk value was assigned based on their risk for causing delays to the projects success and completion. The estimated days for each task's completion is based on working days not calendar days. The chart is ordered so tasks depending on a previous task's completion are started after that task has been completed. Tasks shown in the chart that have not been started but show a progress percentage already are functions utilized in previous labs so a template exists that will later be integrated.

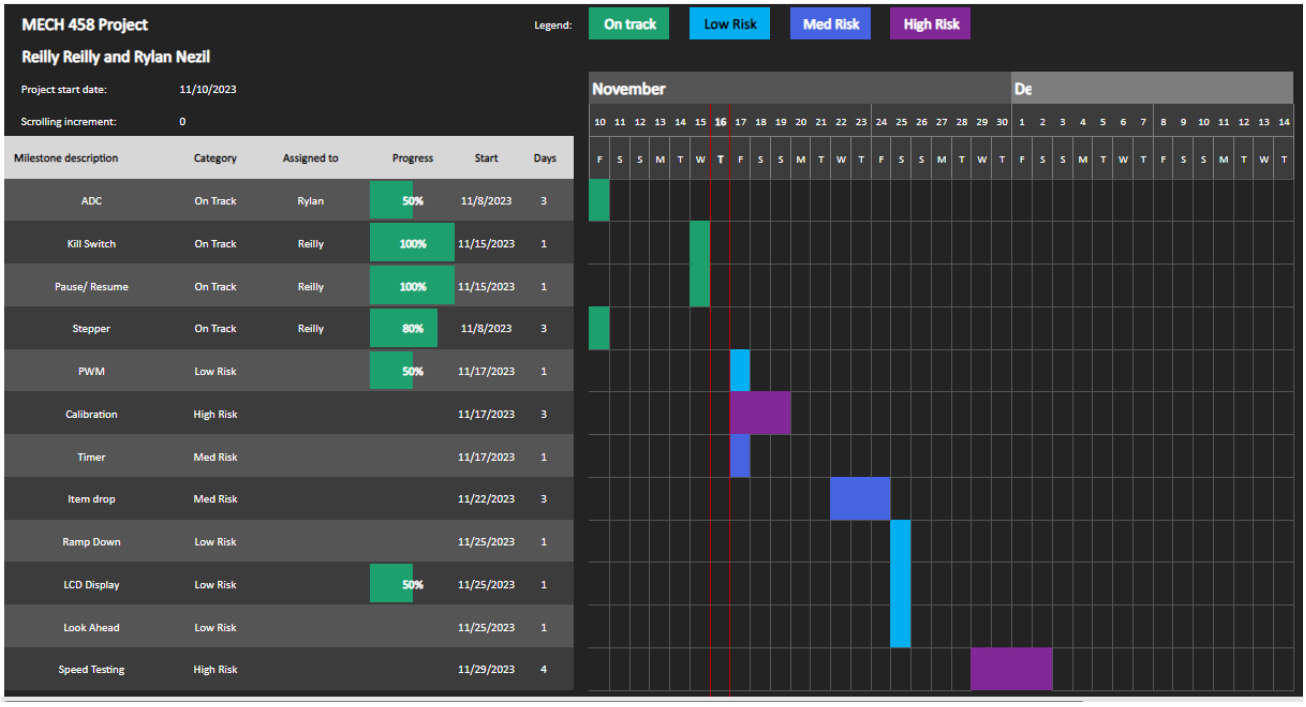

Figure 5. Gantt chart for predicted completion timeline.

3.0 Flow Chart

Figure 6 below shows the flow chart for the sorting machine program:

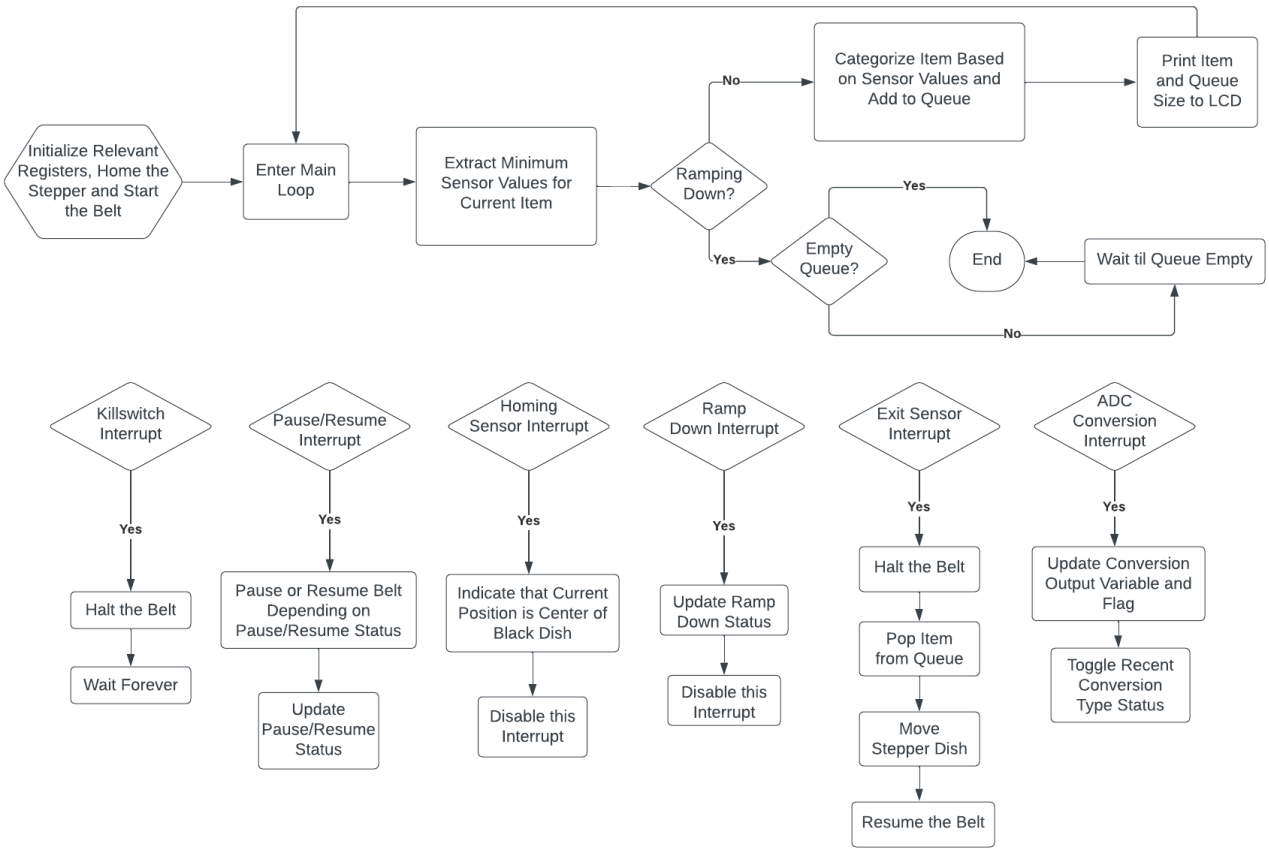

Figure 6. Flow chart for sorting machine program.

4.0 Explanation/Description

The program begins by setting up I/O ports and pins, interrupts, ADC settings, homing the stepper motor, initializing the queue, initializing the LCD, setting up the timer and setting up the PWM. Noteworthy details include of this process include:

- the clock speed is reduced from 16MHz to 8MHz;
- PB1 is used for PWM;
- PORTAL is used for the conveyor belt DC motor;
- PORTA is used for the stepper motor;
- PORTC is used for the LCD display;
- automatic interrupt firing after ADC conversion is enabled;
- free running ADC mode is enabled; and
- PWM frequency is set to 3.9KHz.

Once initialization is complete the main loop is entered, which proceeds as follows:

1. determine the 'peaks' from both the ferromagnetic and reflective sensors for the current item;
2. categorize the item based on these values;
3. add the item to the queue; and
4. print the newly added item to the LCD, along with the current queue size.

Step 1 is the most complicated. Peak determination is accomplished by operating the ADC in free running mode on 2 channels, one corresponding to the ferromagnetic sensor and one corresponding to the reflective sensor. Each time a conversion is completed, the ADC toggles a bit indicating whether the conversion was for a ferromagnetic sensor value or for a reflective sensor value. This bit is then used to toggle the ADC input multiplexer between both sensors, resulting in the conversion output alternating between ferromagnetic sensor values and reflective sensor values. Back in the main loop, values from both sensors are repeatedly compared with the current lowest value for the current item. Once the current values return to no-item values, the minimum value from each sensor is recorded and used to define the item as either steel, aluminium, black plastic or white plastic. Finally, the item is added to the queue and its type is printed to the LCD screen along with the current queue size.

The killswitch interrupt has the highest priority and is triggered by a rising or falling edge. Upon triggering, the belt is stopped and an infinite no-op loop is entered. The microcontroller must be reset in order to resume operation after the killswitch has been thrown.

The pause/resume interrupt has the second highest priority and is triggered by a rising edge. Upon triggering, a bit encoding the current state of the belt is evaluated and the state of the belt is changed accordingly. The bit is then toggled to indicate the new state of the belt.

The homing sensor interrupt is used only once (during the initialization phase of the program) and is triggered by a falling edge. Its purpose is to calibrate the stepper motor such that its 'home' position is located at the center of the black dish. This interrupt disables itself after firing for the first time.

The ramp down interrupt stops new items from being processed, but allows the system to finish sorting the items currently in the queue. It is triggered by a falling edge. This is accomplished by setting a bit indicating that the ramp down state has been entered. This bit is evaluated as a branch condition in the main loop: if the evaluation succeeds, the program waits for the queue to be empty and then exits. Since the operation of the belt and the stepper are controlled by interrupts, the waiting loop does not need to perform any actions other than checking whether the queue is empty or not.

The exit sensor interrupt triggers when an item reaches the end of the belt (falling edge). This interrupt immediately stops the belt, but does not update the belt status. The first item in the queue is popped and the stepper is moved according to the item type. The belt is then resumed.

The ADC conversion interrupt has the lowest priority of all the interrupts, but is triggered the most frequently. Upon triggering, a bit indicating whether the conversion output represents a ferromagnetic sensor value or a reflective sensor value is toggled. This bit allows the main loop to distinguish between these two types of values. The ADC multiplexer MUX0 bit is also toggled (either from Channel 0 to Channel 1 or vice versa), which causes the ADC to perform alternating conversions on ferromagnetic sensor values and reflective sensor values.

5.0 Circuit Diagram

Figure 7 below shows the complete wiring diagram for the system. The pinout is also included below.

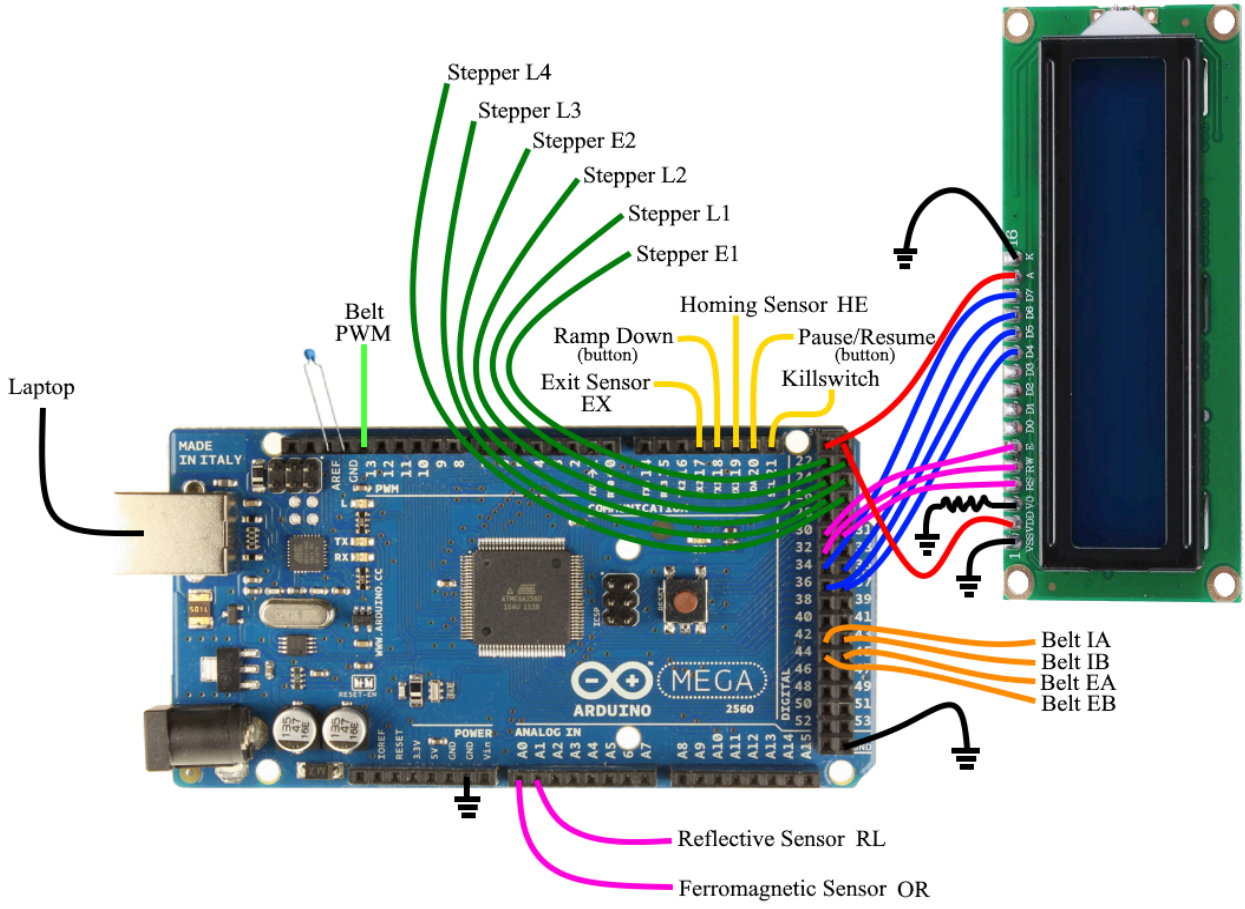

Figure 7. Wiring diagram for the sorting system.

ID	PIN(s)
Belt PWM	13 / PB7
Belt IA	42 / PL7
Belt IB	43 / PL6
Belt EA	44 / PL5
Belt EB	45 / PL4
Killswitch	21 / PD0
Pause/Resume	20 / PD1
Homing Sensor	19 / PD2
Ramp Down	18 / PD3
Exit Sensor	17 / PD4
LCD K	GND
LCD A	VDD
LCD D7:D4	34:37 / PC3:PC0
LCD E	30 / PC7
LCD RW	32 / PC5
LCD RS	31 / PC6
LCD V0	100 Ohm to GND
LCD VDD	VDD
LCD VSS	GND

Stepper E1	22	/	PA0
Stepper L1:L2	23:24	/	PA1:PA2
Stepper E2	25	/	PA3
Stepper L3:L4	26:27	/	PA4:PA5
Ferro Sensor	A0	/	PF0
Reflect Sensor	A1	/	PF1

Appendix B: Detailed Technical Documentation

B.1 High Level System Diagram

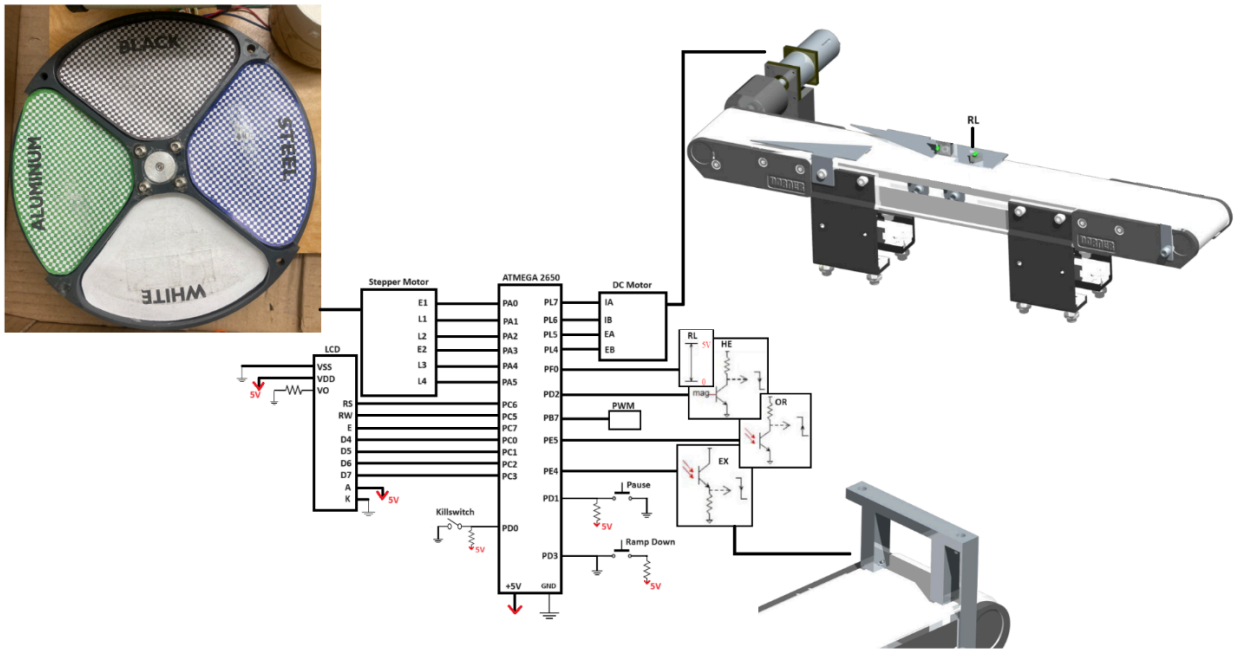

Figure 3.1. High level system diagram.

B.2 Wiring Diagram

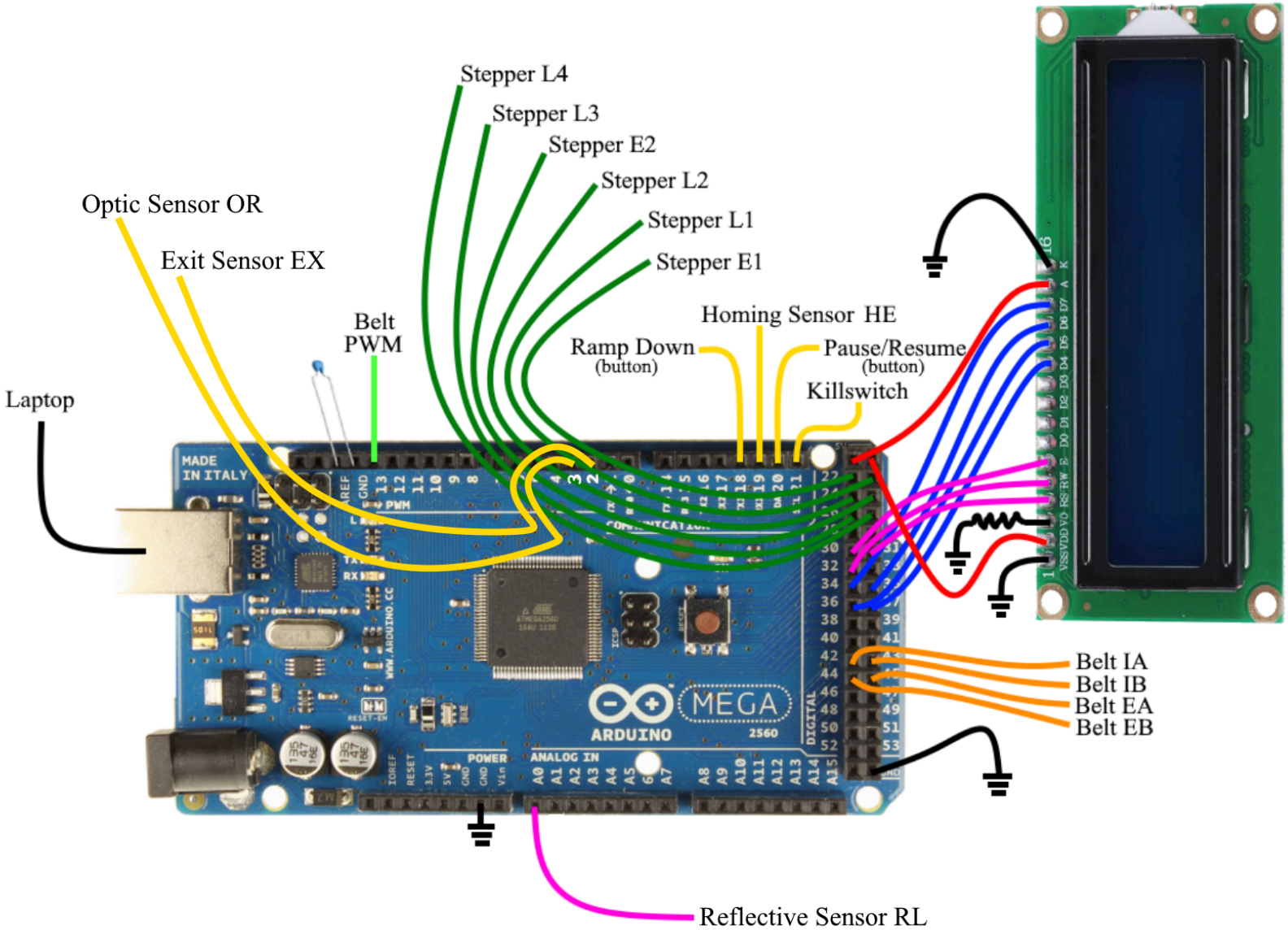

Figure 3.2. Wiring diagram.

B.3 Pinout

PART	BOARD	PORT/PIN
Belt PWM	13	PB7
Belt IA	42	PL7
Belt IB	43	PL6
Belt EA	44	PL5
Belt EB	45	PL4
Killswitch	21	PD0
Pause/Resume	20	PD1
Homing Sensor	19	PD2
Ramp Down	18	PD3
Exit Sensor	2	PE4
Optic Sensor	3	PE5
LCD K	GND	
LCD A	VDD	
LCD D7:D4	34:37	PC3:PC0
LCD E	30	PC7
LCD RW	32	PC5
LCD RS	31	PC6
LCD V0	100 Ohm to GND	

LCD VDD	VDD	
LCD VSS	GND	
Stepper E1	22	PA0
Stepper L1:L2	23:24	PA1:PA2
Stepper E2	25	PA3
Stepper L3:L4	26:27	PA4:PA5
Reflect Sensor	A0	PF0

B.4 Project Code

The project code can be found on Ryland's Github page at the following link.

Appendix C: Summary of Contributions

Reilly:

- Programming and testing of the stepper motor
- Design and implementation of stepper motor functions
- General troubleshooting and testing
- Equal share of circuitry
- Equal share of initial and final reports

Ryland:

- ADC programming
- Interrupt programming
- Timer programming
- Design and implementation of calibration modes
- Design and implementation of main loop
- Design and implementation of system parameters
- Design and implementation of item identification process
- Design and implementation of exit process
- Design and implementation of ramp down process
- Linked list improvement
- General troubleshooting and testing
- Equal share of circuitry
- Equal share of initial and final reports