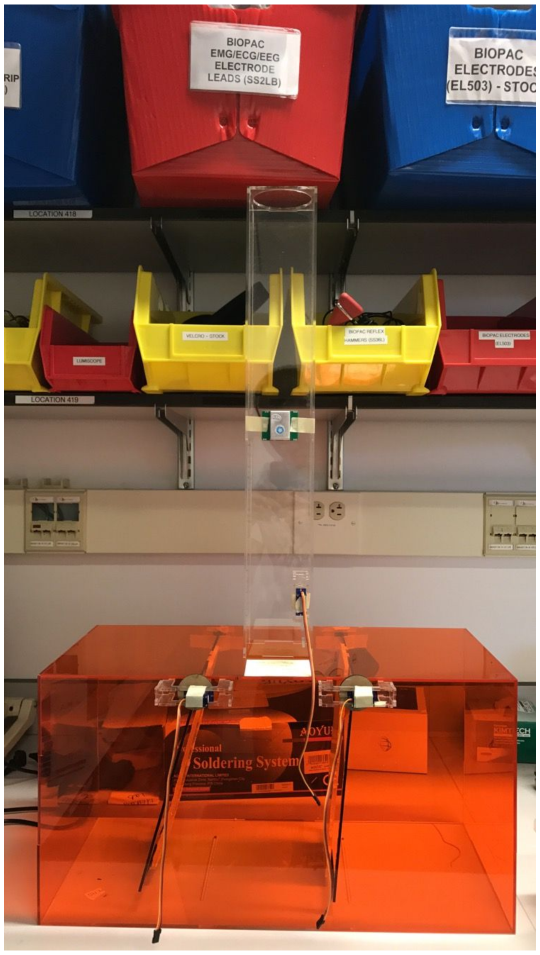
ESE 350 Final Project Report: AutoSort

Frank Fan and William Yang

Background and Problem Statement:

One of the biggest obstacles in recycling, a $500 billion dollar industry, is its high costs. One third of these costs come from the human labor needed to sort and separate different recyclable materials. In attempt to reduce the cost of recycling, we created a way to automate the recycling sorting process upstream at the civilian level.

Overview:



Upon disposal of PET plastic, paper, or aluminum cans, AutoSort sorts and separates the materials. As the trash is dropped through the trash chute, it crosses the pathway between a red LED and photoresistor, generating a unique analog signature that is dependent on the type of recyclable item. The analog signals are read as ADC values on an ATMega328p, which classifies the trash as paper, plastic, or aluminum. Based on the classification, the microcontroller writes a pair of servo motors to move to a certain position. The servo motors control two doors in the bin that direct the trash to secondary bins assigned to contain either plastic, paper, or metal.

Goals:
The goal of the project was to create a system that uses sensors and actuation to perfectly sort recyclable materials into separate bins. We broke this major goal into three milestone subgoals:
- Properly sense and classify the materials
- Actuate a pair of motors to certain positions based on classification
- Integrate all components together in one physical system to meet overall goal

Design Process:
Sensing: We needed the sensor to provide analog outputs dependent on the material of the trash and not the size, shape or weight. We also wanted our final product to have as little moving parts as possible. In order to meet these criteria, we considered radar and optical sensors. Although we determined that the radar sensor was great at distinguishing metals from other materials, we found that it could not output distinct enough analog signals between paper and plastic. On the other hand, using a red LED and a photoresistor gave us distinguishable analog output among all three classes of materials. Thus, we ultimately chose to use the red LED and photoresistor and determined the average ADC readings for each material by repeatedly dropping sample materials through the trash chute in front of the sensor.

Classification: The ATMega328p polls ADC readings and stores these readings in a 40-integer array once the sensor picks up under 5 consecutive ADC measurements above the ambient baseline reading. Once the array is full, the average is taken and compared to threshold ADC values determined from prior experimentation. The microcontroller makes a classification when the average ADC value crosses one of the threshold values.

Actuation: Upon classifying the material, the microcontroller outputs pulse-width modulation signals to control servo motors to move two doors, directing the material to the correct categorized bin.

We chose AutoSort's method of actuation because it was a simple, space-efficient, and time-efficient way of actuation without requiring moving parts in front of our sensor. We also chose this design because it allows gravity to do all the work to move the trash. in addition, the design works well with readily-available servo motors provided by the Bioengineering Lab.

AutoSort's physical body was designed in SolidWorks and built by laser-cutting sheets of acrylic and press-fitting the pieces together.

Results:
AutoSort was able to perfectly sort soda cans of different colors (red and green), plastic water bottles, and paper cups: https://www.youtube.com/watch?v=5Jwo5JaNu5c. Although this serves as a proof of concept, to fully meet our desired goals would require expansion of the different classes of materials that AutoSort can sort. In addition, further tests need to be conducted on

more heterogeneous mixtures of material classes (i.e. sorting newspaper and printer paper into the same bin). Results from sensor testing leading up to our final version of the code are shown below.

Sensing results to obtain threshold ADC values:
https://docs.google.com/spreadsheets/d/1PXvTJWpRYKEZru0hOZCWNEf1yE7adYmFKcd2r9XTi9k/edit?usp=sharing

AutoSort GitHub Repository: https://github.com/pr33ch/AutoSort/blob/master/AutoSort.c (Source code is also pasted at the end of this report)

Challenges

The biggest challenge we ran into was getting consistent sensor output distinct to each material. We tried several analog signal processing methods to make the output of the radar sensor more robust, including signal amplification and high-pass filtering. Despite these efforts, we could not get robust enough output from the sensor. To top it off, the radar sensor ended up breaking at the last minute. In fact, using the LED-photoresistor setup was a last-minute adjustment in response to this. And ironically, this worked greatly in our favor, even simplifying our code.

Another challenge that we faced was building the physical body of AutoSort from scratch. Neither Frank nor I are mechanical engineers and have limited experience with machining and designing physical systems. In addition, designing and building the body was more complicated than expected, especially figuring out how to attach the doors to the motors and the motors to the walls. However, overcoming this challenge was simply a function of time to learn new skills and iterate through different ideas.

Future Improvements:

Future improvements for AutoSort would increase the types of materials that it can sort. The physical size can be scaled up and more experiments to determine average ADC readings can be performed to accommodate more classes of materials. In addition, more sensors such as the radar sensor can be added in series so that sensors can specialize in identifying one type of material. The algorithm can also be improved by looking for more features in addition to the average of the 40 ADC readings. With improved feature selection, machine learning methods can be applied to classify the material.

Source Code:

```
#include "uart.h"
#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#define METAL_THRESHOLD 160
#define PAPER_THRESHOLD 51
#define AIR_THRESHOLD 20

#define PLASTIC_THRESHOLD 740
#define MEASURING_THRESHOLD 840

volatile unsigned int arr[40] = {890, 890, 890, 890, 890, 890, 890, 890, 890,
  890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890,
  890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890, 890};
volatile unsigned int index = 0;
volatile unsigned int prev_adc = 890;
volatile unsigned int flag = 1; //cases in the process of measuring
volatile unsigned int counter = 0;
volatile unsigned int counter_2 = 0;

volatile unsigned int adc_0 = 890; //10 value before
volatile unsigned int adc_1 = 890;
volatile unsigned int adc_2 = 890;
volatile unsigned int adc_3 = 890;
volatile unsigned int adc_4 = 890;

unsigned int r_duty_cycle_2 = 2900; //timer counts for straight, right motor
unsigned int r_duty_cycle_1 = 1400; //timer counts for right
unsigned int r_duty_cycle_3 = 3900; //timer counts for left
unsigned int r_off_cycle_2 = 37100;
unsigned int r_off_cycle_1 = 38600;
unsigned int r_off_cycle_3 = 36100;

unsigned int l_duty_cycle_2 = 2500; //timer counts for straight, left motor
unsigned int l_duty_cycle_1 = 1400; //timer counts for right
unsigned int l_duty_cycle_3 = 3500; //timer counts for left
unsigned int l_off_cycle_2 = 37500;
unsigned int l_off_cycle_1 = 38600;
unsigned int l_off_cycle_3 = 36500;

char HiorLo_R = 0;
char HiorLo_L = 0;

volatile unsigned int duty_period_r;
volatile unsigned int off_duty_period_r;
volatile unsigned int duty_period_l;
volatile unsigned int off_duty_period_l;
```

```
ISR (TIMER1_COMPA_vect) {
  if (HiorLo_R == 0) {
    OCR1A += duty_period_r;
    HiorLo_R = 1;
  }
  else {
    OCR1A += off_duty_period_r;
    HiorLo_R = 0;
  }
}

ISR (TIMER1_COMPB_vect) {
  if (HiorLo_L == 0) {
    OCR1B += duty_period_l;
    HiorLo_L = 1;
  }
  else {
    OCR1B += off_duty_period_l;
    HiorLo_L = 0;
  }
}

int min_of_arr(volatile unsigned int a[], int num_elements)
{
  volatile int tgt = a[0];
  int i;
  for (i=0; i<num_elements - 1; i++)
  {
    if (tgt > a[i+1]) {
      tgt = a[i+1];
    }
  }
  return tgt;
}

void clear_arr(volatile unsigned int a[], int num_elements)
{
  int i;
  for (i=0; i<num_elements; i++)
  {
    arr[i] = 890;
  }
```

```
}

int main() {
  uart_init();

  DDRB |= 0x06; //set pin9 and pin 10 to output
  TCCR1B |= 0x02; //Initialize timer with 8 prescalar
  sei();

  ADCSRA |= 0x07; // ADC prescaler to 1/128 clocks -> 9.6 kHz sampling frequency
  ADMUX |= (1<<REFS0);
  ADMUX &= ~(1<<REFS1); // AREF = 5V
  ADMUX |= (1<<MUX1); //use pin A2
  ADCSRA |= 0x80; // enable ADC

  while (1){
    // printf("%d", flag);
    ADCSRA |= 0x40; // Start an ADC measurement
    while(ADCSRA & 0x40);
    //850 780
    if (counter > 0) {
      counter --;
    }
    else if (ADC > MEASURING_THRESHOLD) {
      if (flag == 0) {
        flag = 1; //start measuring
      }
      else if ((counter == 0) && (flag == 1) && (prev_adc < MEASURING_THRESHOLD)) {
        printf ("it is plastic!!!!!!!!!!!\n");

        flag = 0;
        // index = 0;
        // clear_arr(arr, 40);

        TCCR1A |= 0x50; //enable output compare on OCR1A and OCR1B to toggle
        duty_period_r = r_off_cycle_2;
        duty_period_l = l_off_cycle_2;
        off_duty_period_r = r_duty_cycle_2;
        off_duty_period_l = l_duty_cycle_2;
        OCR1A = TCNT1 + 16; //pull PB1 pin high quickly
        OCR1A += duty_period_r;
        OCR1B = TCNT1 + 16; //pull PB2 pin high quickly
        OCR1B += duty_period_l;
        TIMSK1 |= 0x06; //enable interrupt for OCR1A and OCR1B
      }

      // if (counter > 0) {
      //   counter --;
      // }
    }
    if (flag == 1) {
      if (ADC < MEASURING_THRESHOLD) {
        printf("%d\n", ADC);

        if (ADC < PLASTIC_THRESHOLD) {
          if (counter_2 < 10) {
            counter_2++;
          }

          if (ADC < 680) {
              printf ("it is metal\n");
              flag = 0;

              counter = 15000;
              TCCR1A |= 0x50; //enable output compare on OCR1A and OCR1B to toggle
              duty_period_r = r_off_cycle_1;
              duty_period_l = l_off_cycle_1;
              off_duty_period_r = r_duty_cycle_1;
              off_duty_period_l = l_duty_cycle_1;
              OCR1A = TCNT1 + 16; //pull PB1 pin high quickly
              OCR1A += duty_period_r;
              OCR1B = TCNT1 + 16; //pull PB2 pin high quickly
              OCR1B += duty_period_l;
              TIMSK1 |= 0x06; //enable interrupt for OCR1A and OCR1B
          }
          else if (counter_2 == 10){
              printf ("it is paper\n");
              flag = 0;
              counter_2 = 0;

              counter = 15000;
              TCCR1A |= 0x50; //enable output compare on OCR1A and OCR1B to toggle
              duty_period_r = r_off_cycle_3;
              duty_period_l = l_off_cycle_3;
              off_duty_period_r = r_duty_cycle_3;
              off_duty_period_l = l_duty_cycle_3;
```

```
              OCR1A = TCNT1 + 16; //pull PB1 pin high quickly
              OCR1A += duty_period_r;
              OCR1B = TCNT1 + 16; //pull PB2 pin high quickly
              OCR1B += duty_period_l;
              TIMSK1 |= 0x06; //enable interrupt for OCR1A and OCR1B
          }
          // printf ("it is other\n");
        }
      }
    }
    prev_adc = ADC;
  }
  return 0;
}
```