Group 44 - NEXT GENERATION DIGITAL WASTE TRACKING SYSTEM

Kevin Leung, KiHye Koo, Pyosang Yoo, Vince Hua

University of British Columbia

EECE 409/429/419/439/400/469

April 28, 2015

Group 44 - NEXT GENERATION DIGITAL WASTE TRACKING SYSTEM

Electrical and Computer Engineering Capstone Project

University of British Columbia
collaborating with UBC SEEDS

YingJie (Vince) Hua
KiHye Koo
Kevin Leung
Pyosang Yoo

Instructor: Pieter Botman

TABLE OF CONTENTS

<u>List of Illustrations</u>

Objectives and Scope

We have been building a 'Digital Waste Management System' for our clients Bud Fraser from UBC SEEDS. The goal of this system is to record the weight of several type of waste bins such as garbage and compost from different locations on campus. The system has to be mobile so it can be moved from location to location and be able to distinguish where each weight recording came from in terms of building number, i.e. Marine residence with multiple buildings. From the client's perspective, this system will be used by workers who will determine the correct type and building number of the bin, determine the weight with our system and store that data on some simple format to be analyzed later [1]. Usable as well as a functional system is the main objective of this project.

The scope of our project is to design a whole system that fits the goals set out by our clients. We have determined that the goal for our project is to have a functional system that is mobile, record the weight of bins and store that data. We have designed this functional system which performs the functionalities our clients need but it is still a prototype as there are other steps and improvements to be made before it is a finished product. This will be discussed later in the client handoff section.

As mentioned before, the requirements we have determined are to have a mobile system capable of recording the weight of bins and storing that data in some sort of removable storage. The non-functional requirements include the platform on which the bins are on such as having a ramp for the bins to be rolled on and have a portable system. We have met this requirement by purchasing an industrial floor scale that has the portability, platform and the its weighing functions. It is also very robust as it is designed for industrial use. In addition, an user interface is a necessary requirement to allow users to interact with the system and receive feedback. On the functional requirements side, we have determined that our system must be able to communicate with this floor scale, store the data on a portable storage and receive and output from and to the user. We have designed a Raspberry Pi controller for this requirement. These functional and nonfunctional requirements set out how we wanted to design our system.

We have determined a few constraints to our system and design. The main constraint revolved around the floor scale. The floor scale is expensive and we have acquired the AMS Sustainability Fund to acquire the scale. We have to select the correct scale that would communicate with a device such as the Raspberry Pi and have ramp and portability. We have purchased the scale that has met the budget within the AMS Fund, communicate via RS-232, has ramp, wheels and a handle for portability.

System Design

Our system is designed to fulfill the requirements, constraints and goals of the project. Many sub-components are put together to build a working system. The major components includes the Raspberry Pi controller and the scale. Communication between these two components are important for the proper weight recording for system. The scale comes with the scale indicator which meant this component is added on as a middleman support for the communication between the scale and the Raspberry Pi.

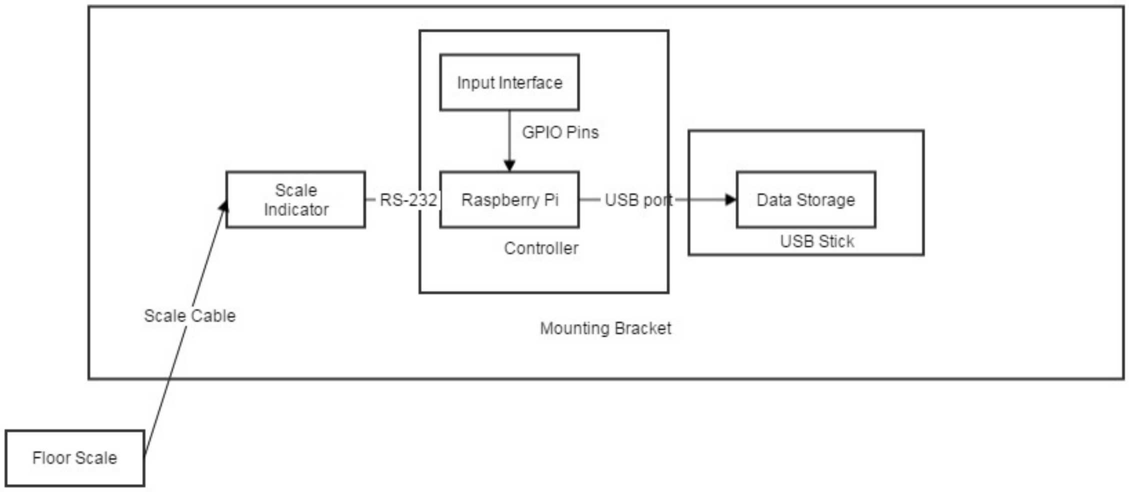


Figure 1. Design Overview

In figure 1, this diagram describes the overall architecture of the system. Physically, the system consists of two parts: the floor scale and the mounting bracket. These components are what the users will carry and use. They are connected via the scale cable provided by the scale manufacture in the box. On the mounting bracket, there are multiple components. The mounting bracket fits the scale indicator, Raspberry Pi controller and the USB stick and it can fit on a waste bin as a stand for the user. Enclosed inside the Raspberry Pi controller is the Raspberry Pi itself and the push button/LED interface circuit connected through GPIO pins to the Raspberry Pi. This module connects to the scale indicator via RS-232 for scale data gathering and to the USB stick via USB port from the Raspberry Pi for data storage.

The role of the floor scale is to provide the platform for the bin and weigh the bins. The scale indicator will read the weight on the scale after receiving the commands on the RS-232 port from the Raspberry Pi controller. The Raspberry Pi controller is where the user interface and the data storage interacts with the user. The user provides inputs to the Raspberry Pi and the Raspberry Pi decides what to do with the inputs and provides outputs depending on what inputs were made. Overall, the system takes input from the Raspberry Pi controller and scale to provide the user with the weight of bins on a data format.

Usability is also a major factor in the design of the system. The scale is portable with wheels and handles as well as a ramp for rolling the bins on. We have combined the scale indicator, Raspberry Pi controller and minimize the amount cables to better portability. The user can carry this set in their hands when moving from location to location. This set is also designed to be put on an empty bin so that the users can stand up and use the system at the same time. It is also angled for better usability. On the user interface side, the user have LED lights providing the feedback for the user and allowing them to push the buttons for the correct settings such as the building number and waste type. The arrangement of the LED lights and buttons are also important factors to consider for a more friendly user experience. There is also a setup stage at the start where user has to put in the waste type and building number before recording the bins as an indication telling the user to setup the system. Error feedback is also included in the system as a blinking red LED light will tell the user that the record is not successful. A error code on the 7-segment display is also provided to tell the user what went wrong in the system. When a user makes a mistake on recording, they can press a delete button which will delete the previous record entry only once to prevent multiple deletions. Multiple recordings on the same bin is also mitigated by allowing another record after the same bin has been removed from the scale. A user guide (Appendix E) is also provided to further maximize the usability of this system. Overall, we have factored in the usability of system and designed the components to provide a good user experience and error recovery when necessary.

Subsystem Design

Scale Platform

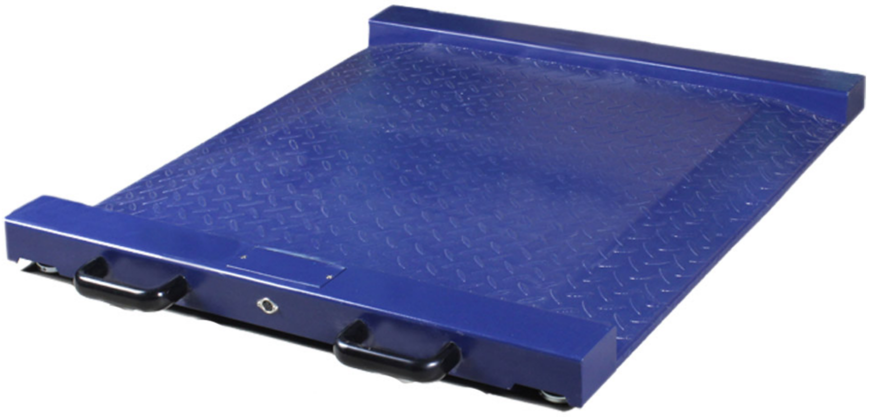


Figure 2. Scale Platform

The industrial floor scale in figure 2 comes with handles and wheels for easy displacement as well as platform for easy bin placement. It has one proprietary output, which connects to the scale indicator. Note that the platform itself does not have any power connector, it relies on the scale indicator for the power delivery. The four feet underneath are the sensor so make sure to handle them with care. The scale also provides a ramp platform to allow bins to be rolled on. This platform provides support for the user so that they can push or pull the bins on and off the scale instead of lifting. The specifications of the scale are in Appendix A.

Scale Indicator

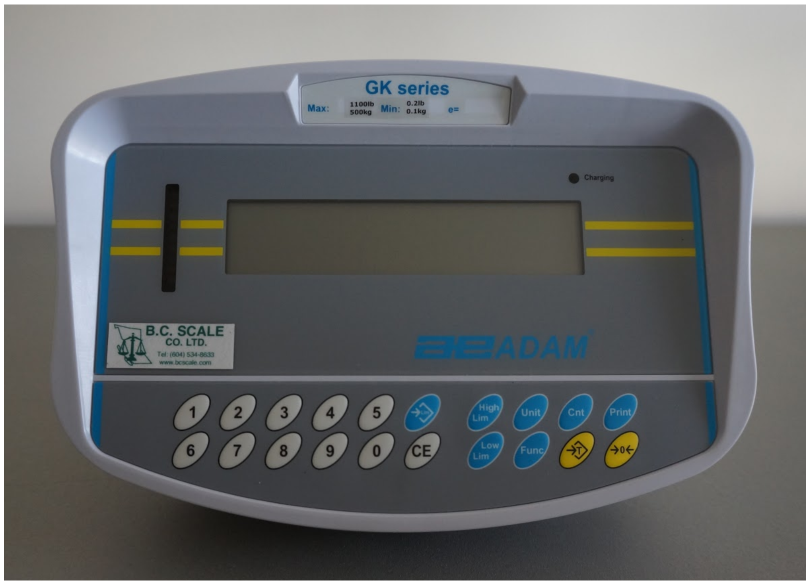


Figure 3. Scale Indicator

As shown in figure 3, the scale indicator is responsible for displaying the weight on the scale platform. It communicates with the scale platform with a proprietary cable. Note that it tares the scale at the indicator's power on, so make sure the scale platform is clear. The indicator comes with a large battery and can be charged with the provided AC adapter. The scale indicator also provides communication to other devices via RS-232. Using our controller fitted with RS-232 communication, we can communicate with this indicator to control it to read the weight on the scale and provide the weight back to the controller. This scale indicator has many buttons and functionalities which come in a separate manual. One of the functionalities the user will use is the tare button. The user has to tare an empty bin first before weighing a number of bins. Most of the other functionalities provided by this scale indicator are setting related.

Controller

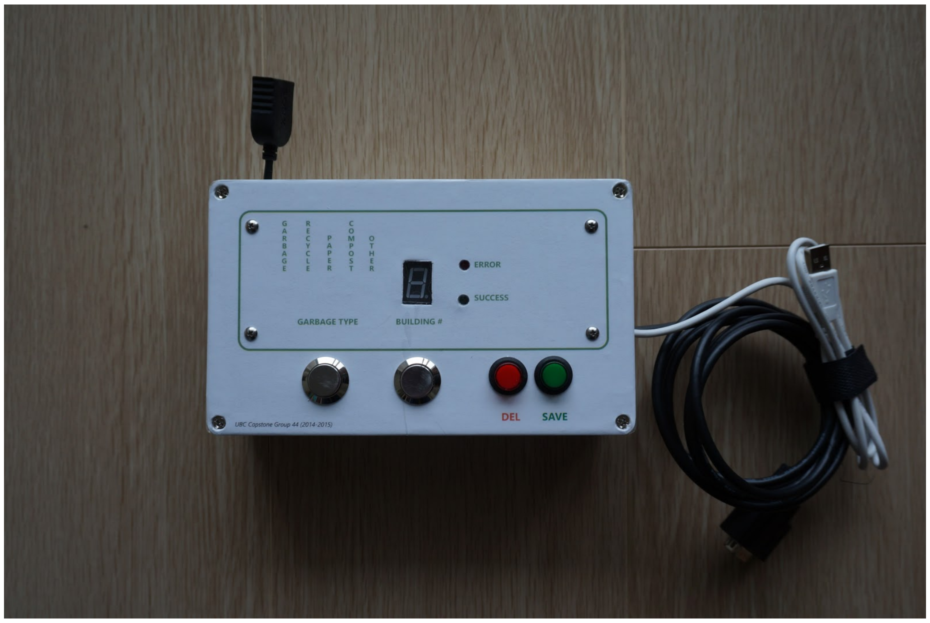


Figure 4. Controller

The controller in figure 4 connects to the indicator with a RS-232 cable. It is responsible for interfacing with the user and output the data to a USB drive. The Raspberry Pi inside runs on 5V and can be powered via usb wall wart, but it can also use a portable PowerBank battery that is capable of powering 5V devices. The Raspberry Pi has the program that runs the system. This program is programmed in Python and is an implementation of the state machine we designed for the system. This controller interacts with the scale indicator through the RS-232 cable and a USB storage device.

The reason on selecting the Raspberry Pi is that it is has the compact size, flexibility and hardware specifications with the pinouts we need. Since we want a compact controller to go along a mobile system, we have selected the Raspberry Pi. The Raspberry Pi's environment is also very flexible for programming and testing. It has many functionalities since it has an operating system. The Raspberry Pi is also capable of communicating through RS-232 serial communication with a shield device that we have purchased. Its environment also allows for better testing and debugging as it is in console environment. The advantage of having an operating system on the Raspberry Pi also provides useful libraries for programming such as RS-232 serial communication protocols and USB storage devices. The GPIO pins from the Raspberry Pi is also very essential to interfacing and powering our controller hardware composed of LEDs and buttons. Overall, the Raspberry Pi has all the tools available to make a functional compact controller.

Inside the controller, there are circuits which are the buttons and LED lights which represent the input and output of the user interface. This circuit is soldered on a PCB and the PCB is bolted on to the back of the front panel. The front panel has holes and is covered with a label for the user interface. The schematics for the circuit are in Appendix D. These are connected through jumper wires to the Raspberry Pi GPIO pins. The mappings from the PCB to the Raspberry Pi GPIO ports as well as the description of the parts are also in the same appendix. This allows for better maintenance as it is simply to connect the jumper wires to the ports. The LEDs and 7-segment display are connected to selected resistor values so that we can limit the amount of current from the GPIO pins from the Raspberry Pi. The Raspberry Pi GPIO pins need to be budgeted to 50mA in total across all GPIO pins with 15mA max for each pin [2]. In addition to the GPIO pins, a RS-232 shield takes up some portion of the GPIO pins for the RS-232 cable connection to the scale indicator. Although it uses some GPIO pins, the shield itself has its own unused GPIO ports and pins that redirects it to the Raspberry Pi's GPIO pins, thus allows us to connect some of the PCB to it. The Python program is also set to these specific ports for the input and output interface. The Raspberry Pi itself is also enclosed and bolted on to the controller enclosure with some holes for external cables such as the RS-232 cable.

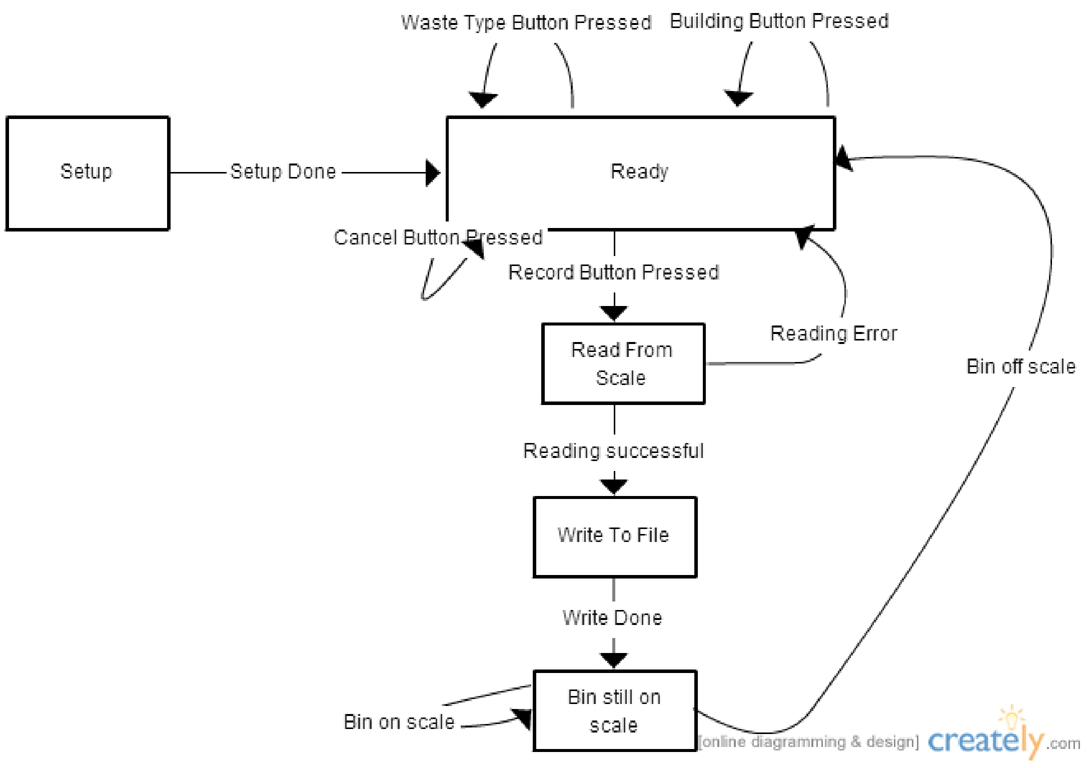


Figure 5. Weight Recording State Machine

Figure 5 represents the state machine of the program on the Raspberry Pi. This state machine is designed for the functionality of the system as well as considering the usability for the user in terms of ease of use and error feedback. In the 'Setup' state is first state in which it takes inputs from the user first before any operation. These inputs are setting up the 'Waste Type' and 'Building Number' for the system before any weight recording. In the 'Ready' state, the controller is ready to poll the scale indicator for the weight. The user presses a 'Save' button on the controller and the controller will get the weight and write to the file to be stored. After that, the user must take the bin off the scale before it is in 'Ready' state again. This prevents multiple records on the same bin, thus increasing the usability of the system. There are also error LEDs that will blink with an error code on the 7-seg display which will tell the user the system has an unsuccessful recording. The Python code implementation of the state machine and overall program is in Appendix B and the details of each state and input and output transaction is in Appendix K.

The formatted data outputted from the file displays a list of all recordings timestamped with a date, bin type and building number. The bin types are ‘Garbage’, ‘Recycle’, ‘Paper’, ‘Compost’ and ‘Others’ and the building number can be between 0-9. The data consists of parts recorded when the controller does not have an USB storage device plugged in and when it does. This allows temporary recordings to be put on the internal storage in form of a SD card on the Raspberry Pi so that it can be later stored onto an USB storage later. The temporary data is deleted from the internal storage once it is loaded onto the USB storage. It will separate the parts with a label and timestamp when the temporary stored data is loaded onto the USB storage. A sample of how the data output looks is in Appendix J.

Mounting bracket

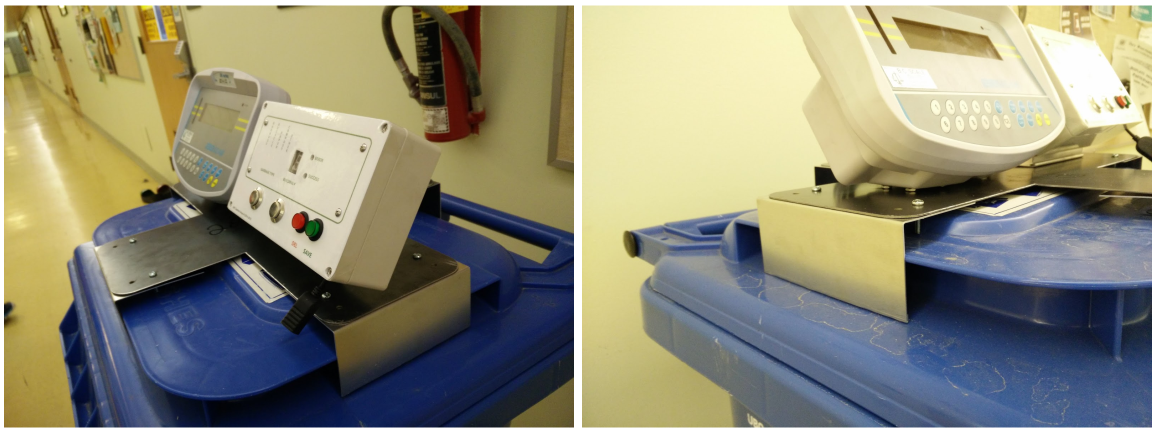


Figure 6. Mounting Bracket on Recycle Bin

The mounting bracket in figure 6 clamps on securely on the garbage bin, It eliminates the need of carrying a separate stand. The notches at the rear of the plate are for cable ties. The four ‘claws’ can be modified if ever the bin’s dimension were to change in the future. There are also two handles conveniently located for easy transportation. As mentioned, this mounting bracket increases the usability by provide these conveniences.

Formal testing

1. User sets up the area, sets up mounting back on an available bin and plugs and powers up controller and scale indicator.
2. User sets up building number and waste type by
3. User places bin on scale.
4. User presses save.
5. User sees success light and removes bin off scale.
6. User repeats process 2-5.

Table 1. Formal Testing

For our testing and verification of our system, we would go through the use case in table 1. We would determine the actual outcome of going through the use case and compare it to the expected outcome. To verify the correct outcome, we would check the output of the user interface such as the LED lights, printouts on console from the Raspberry Pi and the data outputted from the Raspberry Pi. A table of the setup and operation test cases are in Appendix C. This table outlines the steps that replicate a use case, expected and actual outcomes. The scenario of the test would be finding a bin to weigh, weigh the bin and determine the output from the data output file. Along the test, the LED lights will indicate where in the state machine it would be. This user case type of testing shows the correct functionality of the system as well as simulating a real scenario as much as possible.

Other tests include error checking and physical aspects of the system. We would make the system produce an error by not placing a bin on the scale, disconnecting the RS-232 connection and de-stabilizing the scale. These produce two separate errors. One of the errors labelled error code 1 as outlined in the user guide in Appendix E, is disconnection of the RS-232 and instability of the scale while error code 2 is an empty scale. The physical aspects tested are the platform of the bins and the fitting of the mounting bracket on an empty. The platform supports the wheels of the bins and the mounting bracket fits on an empty bin we have found around the labs.

We also have consulted with our clients for a verification test. They would also go through the use case scenario and tell us what they have noticed. They have verified a working functional system but noted that minor issues such as light dimness and delay between button presses which may be a problem for some users [3]. Currently, this issue exists due to the high resistor values mentioned above in the controller section and the program of detecting button inputs. Overall, they have verified it is a functional system suitable for use.

As for tests we are not able to achieve is a full system run where the system is placed in its ideal environment. This test would cover and verify that the system is able to sustain a real scenario as opposed to a few records. Even though our clients have given their test review of the system, we do not know about other user's experience as our clients may be more familiar with our system. Full robustness and sustainability for a long period of time will also test how well built our system is. As mentioned, this system is a prototype which means it does not have the built to sustain like a certified product but it will indicate how reliability the current design is.

Sustainability and Ethics

The next generation digital waste tracking system is designed with sustainability in mind. By monitoring a building's total waste output and the proportion of the different waste types, we can improve vastly its ecological footprint. As informed by our client, they are aiming for 70% of the total waste to be recycled or composted, and this is easily achievable with the device we have manufactured [1]. By the simple push of a button, the weight along with their respective details are recorded onto a text file, which logs the weight output history of a building for further analysis.

On another aspect, we have provided a more sustainable solution to track waste output per building. The portability of the device can avoid the need of having a dedicated unit for each building. One device can monitor multiple building which cuts down both the material used and cost for the manufacturing. Moreover, we have provided to the client circuit diagrams, Solidworks files and item purchase invoices in case if any component were to break or malfunction, extending the product's lifetime.

We have learned that even though our system will be used by our clients, our system is only a prototype which means it is not a finished product to be used like an industrial product. In order to have a complete product, our system must go through CSA certification so that our system can be used by customers with peace in mind knowing that our system is approved for various safety standards [4]. This is critical to engineering as according to the APEGBC Code of Ethics, it is an engineer's duty to consider the safety of everyone [5]. As for our prototype system, we must inform our clients the potential risks involved with using our system.

Project Management and Client Hand-off

We have designed a prototype system for our clients. This prototype does have the functionality that satisfies an use case of weighing, recording, storing and outputting data of the different type of bins with different building number but it is not a final system. To have a final system, the clients or the team after us would have to design a better enclosure or perhaps re-arrange some of the components into a more compact form. It would also have to go through CSA approval to be approved for industrial use. Nevertheless, our clients will use our system in the locations around campus. We will provide them with a user guide (Appendix E) along with this report.

In the beginning, we have planned on implementing a server, mobile phone or email applications as another approach for the data output. Since we have had time constraints, we have not been able to consider these other design options. In the end, we have aimed towards a better built and robust system as opposed to one with more advanced features. This decision gives us a better understanding of how to build a finished product that is able to be used in an industrial setting. Even though our system is a prototype that is not a finished product, we have aimed towards a more robust controller with bolted mounts with organized wiring instead of unmounted and taped up circuits with loose wiring which we have had previously.

Conclusion

Over a course of 8 months, we have developed a fully functional prototype for our clients at UBC SEEDS. We have learned the process of designing this system at different levels. On a high level, we have designed a architectural model with all the interactions of sub-components while considering the goals, requirements and constraints of the system. Selecting the correct equipment such as our scale for our system is essential to work around the requirements, communication protocol and budget constraints. We can then work further down designing each the sub-components such as the Raspberry Pi controller. The Raspberry Pi controller can be modelled using a state machine diagram which can be implemented into a Python coded program. Testing is also needed to verify a working system. We have also learned the sustainability and ethics from our system. Our system is designed for a sustainable cause in hopes of reducing garbage output and better outputs from other types of waste. Since our system is a prototype, we have learned that our system have limitations as it is not a finished product. To reduce these limitations, our system will need further upgrades to meet CSA certification so that it be labelled as a finished project and be used widely.

Reference/ Bibliography

[1] B. Fraser. UBC SEEDS. Meeting Interviews. Vancouver B.C. Sept 2014-March 2015.

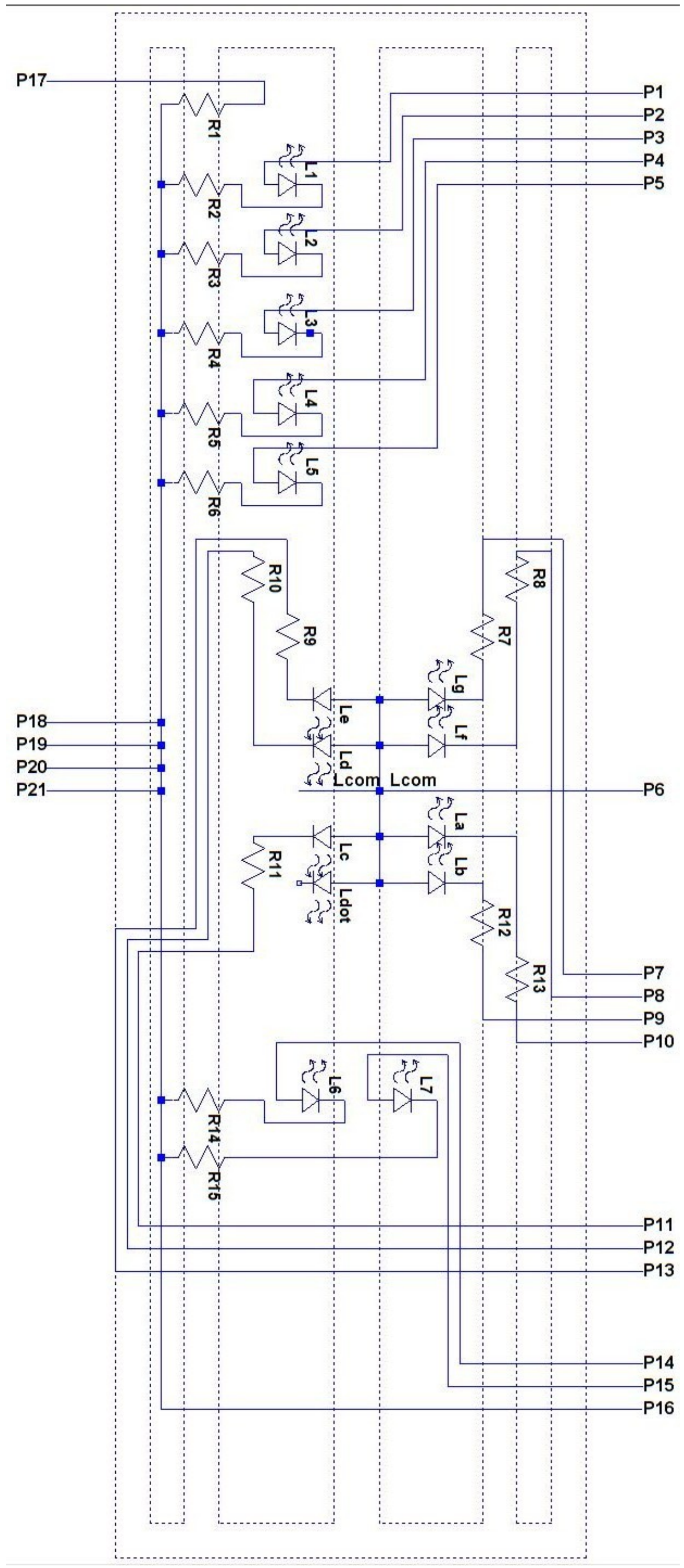
[2] W.W. Gay, *Raspbery Pi Hardware Reference*. Apress, 2014.

[3] B. Fraser and Ivana. UBC SEEDS manager and intern zero waste program coordinator. Testing Review. Vancouver B.C. March 27, 2015.

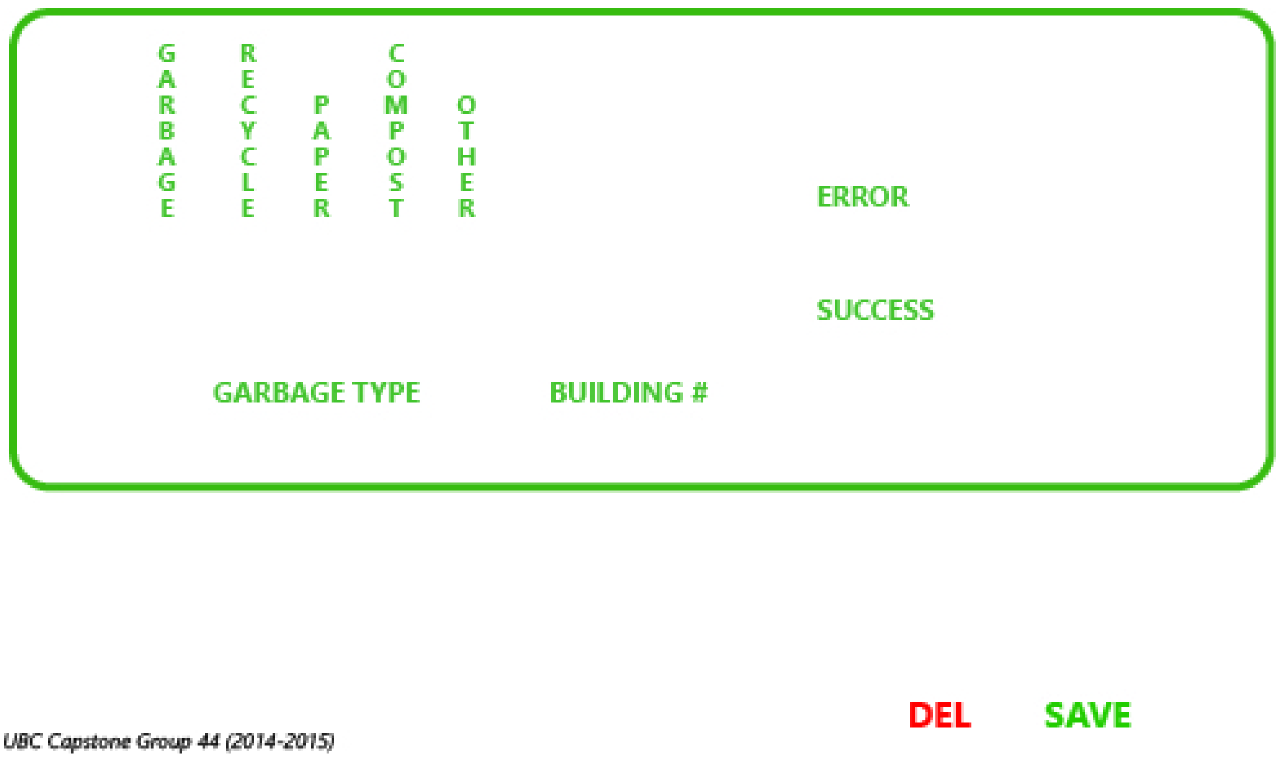
[4] *Why You Need Certification Marks* (n.d.) [Online]. Available: http://www.csagroup.org/ca/en/about-csa-group/certification-marks-labels/why-you-need-certification-marks. Accessed: April 8, 2015.

[5] *APEGBC Code of Ethics* (n.d.) [Online]. Available: https://www.apeg.bc.ca/For-Members/Ethics,-Law-and-Conduct#APEGBCCodeofEthics. Accessed: April 8, 2015.

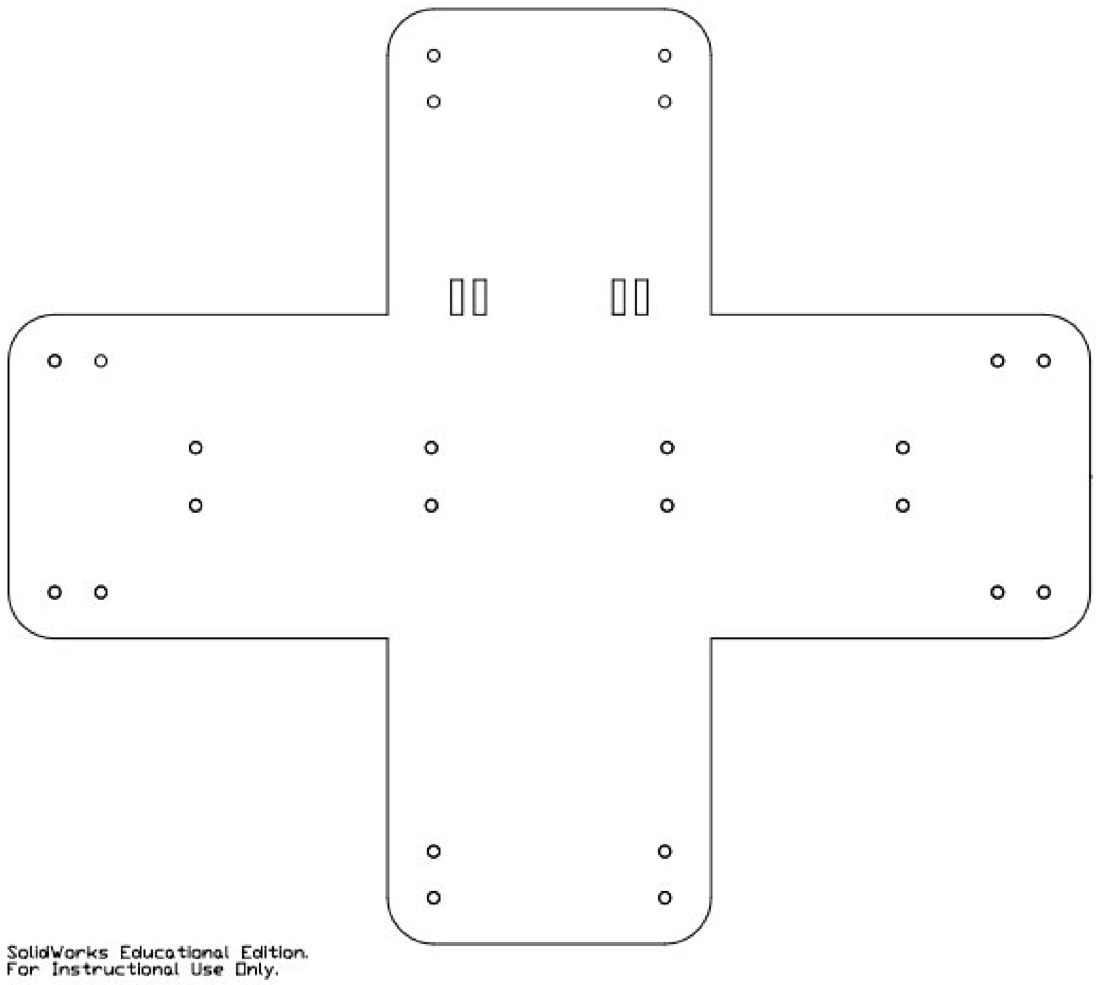
Appendix A - Circuit Diagram



Appendix B - Controller Label



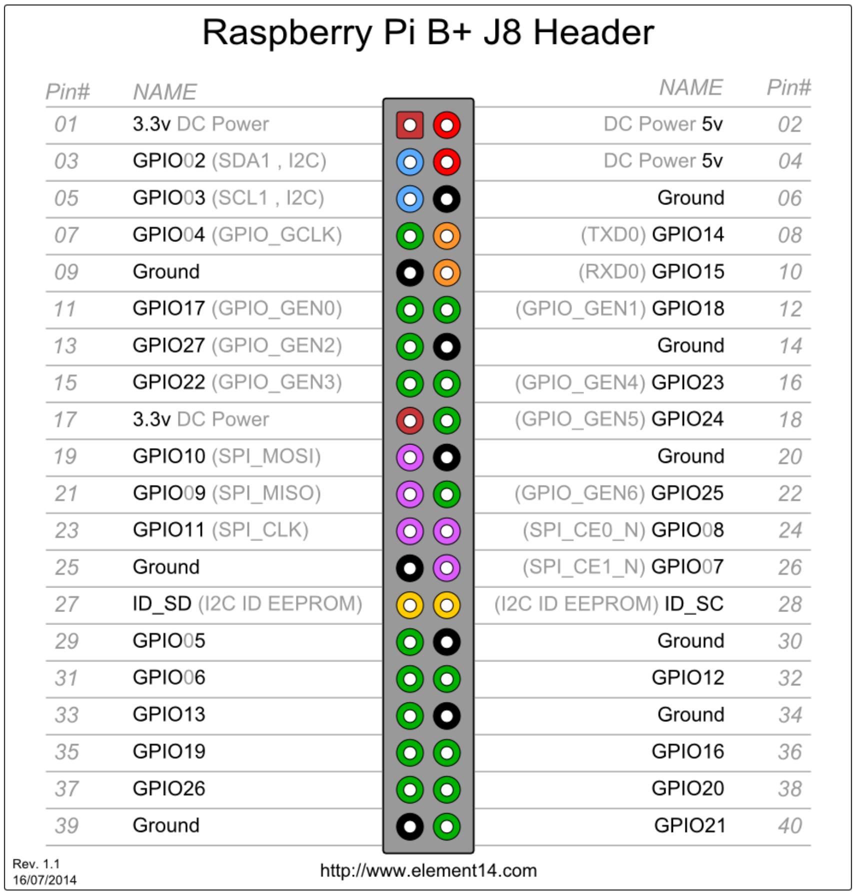
Appendix C: Mounting Bracket



Appendix D: Pinout

PORTS

P1: D7 (labelled on the RS-232 Shield (not shown in this appendix but on top of Raspberry Pi))
P2: D6 (labelled on the RS-232 Shield)
P3: D5 (labelled on the RS-232 Shield)
P4: D4 (labelled on the RS-232 Shield)
P5: D3 (labelled on the RS-232 Shield)
P6: to be connected to the 3.3 volt power port of RS232 SHIELD (3.3)
P7: GPIO 20 (pin 38)
P8: GPIO 16 (pin 36)
P9: GPIO 6 (pin 31)
P10: GPIO 21 (pin 40)
P11: GPIO 19 (pin 35)
P12: GPIO 13 (pin 33)
P13: GPIO 26 (pin 37)
P14: D2 (labelled on the RS-232 Shield)
P15: D1 (labelled on the RS-232 Shield)
P16: this port is to be connected to the ground port of raspberry pi (GND)
P17: Save LED light: D0 (labelled on the RS-232 Shield)
P18: Ground: to one end of one of the four buttons
P19: Ground: to one end of one of the four buttons
P20: Ground: to one end of one of the four buttons
P21: Ground: to one end of one of the four buttons



Raspberry Pi B+ J8 Header

Pin#	NAME			NAME	Pin#
01	3.3v DC Power	🟥	🔴	DC Power 5v	02
03	GPIO02 (SDA1 , I2C)	🔵	🔴	DC Power 5v	04
05	GPIO03 (SCL1 , I2C)	🔵	⚫	Ground	06
07	GPIO04 (GPIO_GCLK)	🟢	🟠	(TXD0) GPIO14	08
09	Ground	⚫	🟠	(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)	🟢	🟢	(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)	🟢	⚫	Ground	14
15	GPIO22 (GPIO_GEN3)	🟢	🟢	(GPIO_GEN4) GPIO23	16
17	3.3v DC Power	🔴	🟢	(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)	🟣	⚫	Ground	20
21	GPIO09 (SPI_MISO)	🟣	🟢	(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)	🟣	🟣	(SPI_CE0_N) GPIO08	24
25	Ground	⚫	🟣	(SPI_CE1_N) GPIO07	26
27	ID_SD (I2C ID EEPROM)	🟡	🟡	(I2C ID EEPROM) ID_SC	28
29	GPIO05	🟢	⚫	Ground	30
31	GPIO06	🟢	🟢	GPIO12	32
33	GPIO13	🟢	⚫	Ground	34
35	GPIO19	🟢	🟢	GPIO16	36
37	GPIO26	🟢	🟢	GPIO20	38
39	Ground	⚫	🟢	GPIO21	40

Rev. 1.1
16/07/2014

http://www.element14.com

RST (resistors)

<u>5 current limiting resistors for the 5 LEDs; L1, L2, L3, L4, L5</u>
R1: resistor for P17, Save LED light (resistance value = 220 Ω) *values are approximate
R2: resistor for L1 (resistance value = 220 Ω)
R3: resistor for L2 (resistance value = 220 Ω)
R4: resistor for L3 (resistance value = 220 Ω)
R5: resistor for L4 (resistance value = 220 Ω)
R6: resistor for L5 (resistance value = 220 Ω)

<u>7 current limiting resistors for the 7 segment display LEDs; La, Lb, Lc, Ld, Le, Lf, Lg</u>
R7: resistor for L7 (resistance value = 2700 Ω) *values are approximate
R8: resistor for P8 (resistance value = 2700 Ω)
R9: resistor for S3 (resistance value = 2700 Ω)
R10: resistor for S4 (resistance value = 2700 Ω)
R11: resistor for S5 (resistance value = 2700 Ω)
R12: resistor for S6 (resistance value = 2700 Ω)
R13: resistor for S8 (resistance value = 2700 Ω)

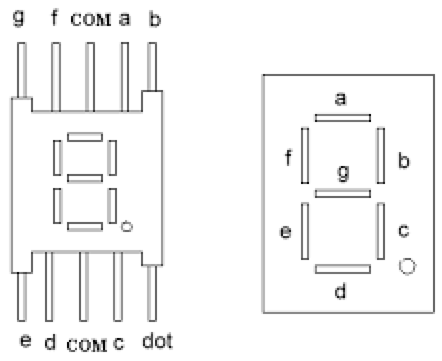
2 current limiting resistors for the 2 LEDs; L6, L7
R14: resistor for L6 (resistance value = 1000 Ω) *values are approximate
R15: resistor for L7 (resistance value = 1000 Ω)

LED (light emitting diodes)
L1: LED for indicating “GARBAGE”, a type of Waste
L2: LED for indicating “RECYCLE”, a type of Waste
L3: LED for indicating “PAPER”, a type of Waste
L4: LED for indicating “COMPOST”, a type of Waste
L5: LED for indicating “OTHERS”, a type of Waste
L6: LED for indicating “SUCCESS”
L7: LED for indicating “ERROR”

SSD (seven segment display LEDs)
Lg: to be connected to “g” port of seven segment display component (P10)
Lf: to be connected to “f” port (P8)
Lcom: to be connected (P6)
La: to be connected to “a” port (P12)
Lb: to be connected to “b” port (P9)
Le: to be connected to “e” port (P)
Ld: to be connected to “d” port (P13)
Lc: to be connected to “c” port (P11)
Ldot: unconnected

Push Buttons (one end)

Save: MOSI (labelled on the RS-232 Shield)
Del: MISO (labelled on the RS-232 Shield)
Waste Type: CE1 (labelled on the RS-232 Shield)
Building Number: CE0 (labelled on the RS-232 Shield)



Seven-Segment Display

Appendix E: Raspberry Pi code

```
import RPi.GPIO as GPIO
import time
import os
import sys
import serial
import signal
import thread
import threading
from enum import Enum

class State(Enum):
    Setup = 0
    Ready = 1
    Read_From_Scale = 2
    Confirm = 3
    Write_To_File = 4
    BinOnScale = 5

def readport(port):
    msg = ""
    while True:
        c = port.read();
        if c != '\r' and c != '\n':
            msg += c
        if c == '':
            print(msg)
            return msg

def talkToScale(port):
    port.write("P\r\n")
    recordWeight = readport(port)
    return recordWeight

def clearleds(leds):
    for led in leds:
        GPIO.output(led,0)
        return

def writeTypeToFile(waste_file, weight, waste_type, building):
    waste_file.write("Time: " + time.asctime(time.localtime(time.time())) + " Type: ")
    localtime = time.asctime(time.localtime(time.time()))
    if waste_type == led_garbage:
        waste_file.write("Garbage")
        print("Garbage wrote to file")
    elif waste_type == led_paper:
        waste_file.write("Paper")
        print("Paper wrote to file")
    elif waste_type == led_recycle:
        waste_file.write("Recycle")
        print("Recycle wrote to file")
    elif waste_type == led_compost:
        waste_file.write("Compost")
        print("Compost wrote to file")
    elif waste_type == led_extra:
        waste_file.write("Others")
        print("Others wrote to file")
    print(localtime)
    waste_file.write(' Weight: {0} kg Building: {1} \r\n'.format(weight, building))

def printType(waste_type):
    if(waste_type == led_garbage):
        print('Type: Garbage')
    elif(waste_type == led_compost):
        print('Type: Compost')
    elif(waste_type == led_paper):
        print('Type: Paper')
    elif(waste_type == led_recycle):
        print('Type: Recycle')
    elif(waste_type == led_extra):
        print('Type: Others')

#Blink error LED and error code on 7-seg for around 5s
def blinkErrorLED(error_code):
    GPIO.output(seg_8, 1);

    n = 10
    GPIO.output(led_error, 1)
    GPIO.output(seg_8,1)
    alt = 1
    while(n>0):
        time.sleep(0.5)
        GPIO.output(led_error, alt)
        if(alt):
            GPIO.output(seglist[error_code], 0);
        else:
            GPIO.output(seg_8, 1);
        n = n - 1
        alt = not alt
    GPIO.output(led_error, 0)
    GPIO.output(seg_8, 1);

#Close ports and GPIO on Ctrl-C (program exit)
def signal_handler(signal, frame):
    GPIO.cleanup()
    port.close()
    print('Exited')
    sys.exit(0)

def timeoutHandler(signum, frame):
    print("Ready state timed out")
    global timeout_flag
    timeout_flag = 1

def scrollLED():
    for i in typelist:
        GPIO.output(typelist,0)
        GPIO.output(i,1)
        time.sleep(0.35)
def scrollSevenSeg():
    for i in segsetup:
        GPIO.output(seg_8, 1)
        GPIO.output(i, 0)
        time.sleep(0.30)
    pass

GPIO.setmode(GPIO.BOARD)

D0 = 11
D1 = 12
D2 = 13
D3 = 15
D4 = 16
D5 = 18
D6 = 22
D7 = 7
MOSI = 19
```

```
MISO = 21
CE0 = 24
CE1 = 26

La = 40
Lb = 31
Lc = 37
Ld = 35
Le = 33
Lf = 36
Lg = 38

led_garbage = D7
led_compost = D4
led_paper = D5
led_recycle = D6
button_select = CE1
button_building_select = CE0
button_record = MOSI
button_cancel = MISO
led_extra = D3
led_ready = D0
led_error = D1
led_success = D2

seg_0 = (La,Lb,Lc,Ld,Le,Lf)
seg_1 = (Lb,Lc)
seg_2 = (La,Lb,Ld,Le,Lg)
seg_3 = (La,Lb,Lc,Ld,Lg)
seg_4 = (Lb,Lc,Lf,Lg)
seg_5 = (La,Lc,Ld,Lf,Lg)
seg_6 = (La,Lc,Ld,Le,Lf,Lg)
seg_7 = (La,Lb,Lc)
seg_8 = (La,Lb,Lc,Ld,Le,Lf,Lg)
seg_9 = (La,Lb,Lc,Lf,Lg)
seg_e = (La,Ld,Le,Lf,Lg)
seg_t = (Ld,Le,Lf,Lg)
seg_u = (Lc,Ld,Le)
seg_p = (La,Lb,Le,Lf,Lg)

seglist = [seg_0, seg_1, seg_2, seg_3, seg_4, seg_5, seg_6, seg_7, seg_8, seg_9]
segsetup = [La,Lb,Lc,Ld,Le,Lf]
```

```
GPIO.setup(led_garbage, GPIO.OUT)
GPIO.setup(led_compost, GPIO.OUT)
GPIO.setup(led_paper, GPIO.OUT)
GPIO.setup(led_recycle, GPIO.OUT)
GPIO.setup(led_extra, GPIO.OUT)

GPIO.setup(button_select, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(button_building_select, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(button_record, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(button_cancel, GPIO.IN, pull_up_down=GPIO.PUD_UP)

GPIO.setup(led_ready, GPIO.OUT)
GPIO.setup(led_error, GPIO.OUT)
GPIO.setup(led_success, GPIO.OUT)

GPIO.setup(La, GPIO.OUT)
GPIO.setup(Lb, GPIO.OUT)
GPIO.setup(Lc, GPIO.OUT)
GPIO.setup(Ld, GPIO.OUT)
GPIO.setup(Le, GPIO.OUT)
GPIO.setup(Lf, GPIO.OUT)
GPIO.setup(Lg, GPIO.OUT)

typelist = (led_garbage, led_recycle, led_paper, led_compost, led_extra)
typecount = 0
TYPECOUNT_MAX = 5
building = 0
BUILDING_MAX = 9
select_pressed = False
record_pressed = False

clearleds(typelist)

state = State.Setup
previous_file = ""
cancel = 0
weight = 0.0
read_flag = 0
timeout_flag = 0
active_flag = 0
scrollThread = None
scrollSegThread = None
```

```
port = serial.Serial("/dev/ttyAMA0", baudrate=9600, timeout=3.0)
port.open()
scale_error = 0

signal.signal(signal.SIGINT, signal_handler)
GPIO.output(seg_8,1)
GPIO.output(typelist,0)
GPIO.output(led_success,0)
GPIO.output(led_error,0)
GPIO.output(led_ready,0)

#determine where to write the file: either locally or on usb
filepath = "/home/pi/dwms/"

if(os.path.ismount("/home/pi/usb/")):
   usbpath = "/home/pi/usb/"
   try:
      recordfile = open(os.path.abspath("/home/pi/dwms/") + "/sample.txt","r")
      record = recordfile.read()

      if(len(record) == 0):
         print('nothing to record')
      else:
         print("doing this")
         usbfile = open(os.path.abspath(usbpath) + "/sample.txt","a+")
         usbfile.write("\r\n**************Previous unstored records**************\r\n")
         usbfile.write("Time stored: " + time.asctime(time.localtime(time.time())) + "\r\n\r\n")
         usbfile.write(record)
         usbfile.write("\r\n*****************************************************\r\n")
         usbfile.close()
      recordfile.close()
      os.remove("/home/pi/dwms/sample.txt")
   except:
      print("file fail")
      pass

   filepath = usbpath

#start of the state machine
while True:
   if(state == State.Setup):
      GPIO.output(led_ready, 0)
```

```
print("Setup State")
print("7-seg displays an \'s\'")
typecount = TYPECOUNT_MAX - 1
building = BUILDING_MAX

flag_type_select = 0
flag_building_select = 0

while True:
    if(GPIO.input(button_select) == False):
        flag_type_select = 1

        if(scrollThread != None):
            while(scrollThread.isAlive()):
                pass

        GPIO.output(typelist, 0)
        if typecount == TYPECOUNT_MAX - 1:
            typecount = 0
        else:
            typecount += 1
        GPIO.output(typelist[typecount], 1)
        printType(typelist[typecount])

        time.sleep(0.5)
        while(GPIO.input(button_select) == False):
            pass
        time.sleep(0.5)
    elif(GPIO.input(button_building_select) == False):
        flag_building_select = 1
        if building == BUILDING_MAX:
            building = 0
        else:
            building += 1
        print("Building: {0}".format(building))

        if(scrollSegThread != None):
            while(scrollSegThread.isAlive()):
                pass

        GPIO.output(seg_8,1)
        GPIO.output(seglist[building],0)
```

```
            time.sleep(0.5)
            while(GPIO.input(button_building_select) == False):
                pass
            time.sleep(0.5)

        if(flag_type_select and flag_building_select):
            state = State.Ready
            break

        if(not flag_type_select):
            if(scrollThread == None or not scrollThread.isAlive()):
                scrollThread = threading.Thread(target = scrollLED)
                scrollThread.start()

        if(not flag_building_select):
            if(scrollSegThread == None or not scrollSegThread.isAlive()):
                scrollSegThread = threading.Thread(target = scrollSevenSeg)
                scrollSegThread.start()

elif(state == State.Ready):

    if scale_error != 0:
        blinkErrorLED(scale_error)
        scale_error = 0;

    read_flag = 0
    timeout_flag = 0
    alarm_flag = 0

    GPIO.output(led_ready, 1);

    GPIO.output(seg_8,1)
    GPIO.output(seglist[building],0)

    print('Ready State')
    printType(typelist[typecount])
    print("Building: {0}".format(building))

    GPIO.output(typelist[typecount], 1)

    #polling for input
    while True:
        if(alarm_flag == 0):
```

```
    signal.signal(signal.SIGALRM, timeoutHandler)
    signal.alarm(120)
    alarm_flag = 1
elif(timeout_flag == 1):
    state = State.Setup
    break

if(GPIO.input(button_record) == False):
    signal.alarm(0)
    state = State.Read_From_Scale
    while(GPIO.input(button_record) == False):
         pass
    break
elif(GPIO.input(button_cancel) == False):
    active_flag = 1
    #allow only one cancel per scale read
    if(cancel == 1):
        print(previous_file)
        file = open(os.path.abspath(filepath) + "sample.txt", "w+")
        file.write(previous_file)
        file.close()
        cancel = 0
        GPIO.output(led_success, 1)
        time.sleep(3)
        GPIO.output(led_success, 0)
elif(GPIO.input(button_select) == False):
    signal.alarm(120)

    GPIO.output(typelist, 0)
    if typecount == TYPECOUNT_MAX - 1:
        typecount = 0
    else:
        typecount += 1
    GPIO.output(typelist[typecount], 1)
    printType(typelist[typecount])

    time.sleep(0.5)
    while(GPIO.input(button_select) == False):
        pass
    time.sleep(0.5)
elif(GPIO.input(button_building_select) == False):
    signal.alarm(120)
```

```
            if building == BUILDING_MAX :
                building = 0
            else:
                building += 1
            print("Building: {0}".format(building))
            GPIO.output(seg_8,1)
            GPIO.output(seglist[building],0)

            time.sleep(0.5)
            while(GPIO.input(button_building_select) == False):
                pass
            time.sleep(0.5)
elif(state == State.Read_From_Scale):
    GPIO.output(led_ready, 0);
    print('Reading from scale')
    scale_message = talkToScale(port)
    if scale_message == "":
        print('Error reading from scale')
        state = State.Ready
        scale_error = 1
    else:
        for s in scale_message.split():
            try:
                print(s)
                weight = float(s)
            except ValueError:
                pass

        #checks if first read or polling to check if bin still on scale
        if read_flag:
            state = State.BinOnScale
        else:
            state = State.Write_To_File
    read_flag = 1
elif(state == State.Write_To_File):
    print('Writing to file')

    if(weight < 2.0):
        state = State.Ready
        scale_error = 2
    else:
        try:
            file = open(os.path.abspath(filepath) + "/sample.txt", "r")
```

```
                previous_file = file.read()
                file.close()
            except IOError:
                print("file does not exist")

            file = open(os.path.abspath(filepath) + "/sample.txt", "a+")
            writeTypeToFile(file, weight, typelist[typecount], building)
            file.close()

            GPIO.output(led_success,1)
            time.sleep(3)
            GPIO.output(led_success,0)

            cancel = 1
            state = State.BinOnScale
    elif(state == State.BinOnScale):
        print("Bin still on scale")
        print(weight)
        if(weight > 1.0 ):
            state = State.Read_From_Scale
        else:
            state = State.Ready
    else:
        state = State.Ready

GPIO.cleanup()
port.close()
```

Appendix F: Sample Data Output

```
**************Previous unstored records**************
Time stored:Thu Mar 19 15:33:38 2015

Time: Thu Mar 19 15:31:48 2015 Type: Garbage Weight: 63.1 kg Building: 0

****************************************************

**************Previous unstored records**************
Time stored: Fri Mar 20 17:42:41 2015

Time: Thu Mar 19 15:50:08 2015 Type: Recycle Weight: 13.0 kg Building: 1

****************************************************

**************Previous unstored records**************
Time stored: Tue Mar 24 18:29:00 2015

Time: Tue Mar 24 18:33:32 2015 Type: Recycle Weight: 13.4 kg Building: 1
Time: Tue Mar 24 18:37:59 2015 Type: Recycle Weight: 13.3 kg Building: 1
Time: Tue Mar 24 18:40:10 2015 Type: Recycle Weight: 18.6 kg Building: 1
Time: Tue Mar 24 18:37:02 2015 Type: Garbage Weight: 13.8 kg Building: 7
Time: Tue Mar 24 18:38:49 2015 Type: Garbage Weight: 13.8 kg Building: 7
Time: Tue Mar 24 18:39:35 2015 Type: Garbage Weight: 13.8 kg Building: 7
Time: Tue Mar 24 18:40:11 2015 Type: Garbage Weight: 13.8 kg Building: 7
Time: Tue Mar 24 18:49:26 2015 Type: Garbage Weight: 70.1 kg Building: 1

****************************************************
Time: Fri Mar 27 15:31:48 2015 Type: Paper Weight: 61.3 kg Building: 2
```

Appendix G: State Machine Diagram

Current State	Input	Output	Next State
Setup	Building selected and type selected	Ready LED ON	Ready
Ready	2-3 min timeout	Ready LED OFF	Setup
Ready	Record Button Pressed	Ready LED OFF	Read From Scale
Ready	Cancel Button Pressed, Cancel = 1	One line is removed from file, cancel = 0	Ready
Ready	Waste type button pressed	Waste type LEDs rotate once	Ready
Ready	Building button pressed	Building indicator increase by one (7-seg display)	Ready
Read From Scale	Reading Error OR Weight < 1kg	Ready LED ON, Error LED ON for 5s	Ready
Read From Scale	Reading Successful		Confirm
Write To File	Write Done	Ready LED ON, file written with new entry, previous file record, cancel = 1	Bin on scale
Bin on scale	Bin off scale (when weight drops near 0)		Ready
Bin on scale	Bin remain on scale	Continue polling	Bin on scale

Note: Cancel button only removes only one of the previous entry which means it will only cancel the last record. Cancel feature will be available again after a new record.

Appendix H: Usability Test

SETUP TEST

First, Identify on the controller where the “Building Number” Push Button, “Waste Type” Push Button, . Also take note of the locations of the set of LEDs “Garbage”, “Recycle”, “Paper”, “Compost”, “Others”, and the “Ready/Record” LED/Button. Then, proceed with the following steps:

User Action	**Expected Outcome**	**Actual Outcome**
1. Press “Building Number” Push Button	Changes Value displayed on the 7 segment display “Building Number”. Scrolls through 0-9. Some delay due to software change to prevent multiple scrolling from one press.	As expected except there’s some delays between presses. Delay between presses are apparent.
2. Press “Waste Type” Push Button	Causes light displayed on one of the LEDs “Garbage”, “Recycle”, “Paper”, “Compost”, “Others” to switch through them.	Similar to test case 1.
3. Mounting bracket on a waste bin	Fits on blue paper waste bin.	Fits on the blue paper bin which the bracket measurements was based on. Not tested on other types.
4. Plug in controller and scale indicator	Scale indicator displays a weight and controller has scrolling LEDs and 7-segment display	As expected. Startup time takes a while after plugging in.

OPERATION TEST

First, identify where the Scale and Indicator are. Also identify the location of the “Cancel” Push Button on the controller. Prepare a sample Garbage Bin with a certain mass of waste inside in. Then, proceed with the following steps:

User Action	**Expected Outcome**	**Actual Outcome**
1. Place Garbage Bin on Scale	Indicator displays weight of waste	As expected
2. Press “Ready/Record” LED/Button	“Save” LED/Button turns off	“Save” LED/Button is turns off. Also it is quite dim to see.
3. Wait Until “Success”	“Success” LED turns on.	“Success” LED turns on for a

LED turns on	Data is recorded successfully	few seconds then off again.
4. Remove Garbage Bin from Scale	“Ready/Record” LED/Button turns on. Operation is successfully completed	If done quickly, “Ready/Record” LED/Button turns on. If the scale indicator is not steady, error light will blink but it does not have any errors.
5. Press “Cancel”	Data recorded from previous operation is deleted	Data recorded from previous operation is deleted. Does this only once per operation. Success LED lit up to show this.

Test Verification with Clients (Bud Fraser and Ivana)

Verified Functions:

- Rolling bins on scale
- User inputs for controller such as “Building Number” and “Garbage Type”.
- Tare on scale indicator
- Confirmations for records in forms of LEDs
- Written to USB and shown on laptop
- Prevent recording nothing on scale
- Error feedback for unstable recordings
- Mountable bracket

Comments:

- Noted that our LEDs were quite dim but bright enough to see for the average user
- Labels with “Garbage Type” may be misleading. Should be “Waste Type”.
- Uncontrollable beeps from the scale indicator - which are now disabled
- Buttons have a little lag between selection
- Good mounting fit with the blue paper bins
- Short confirmation on successful records - now fixed by extending delay

Appendix I: User Guide

Digital Waste Management System

User Guide

UBC Capstone 2014-2015

By: YingJie (Vince) Hua, KiHye (Shelley) Koo, Kevin Leung, Pyosang (Peter) Yoo

Table Of Contents

Disclaimer

This device is not CSA certified thus does not have the certification to be have industrial workplace standards. This device is a prototype that does the functions as programmed but as a prototype, it may not pass CSA standards, according to a safety review due to its enclosure built and internal wiring. Documentation is provided along for the detailed technical specification of this device for any future design upgrades. The Raspberry Pi in the controller uses 5V which is not hazardous but due to its built, use this device with caution.

Components

Scale Platform

Description: The scale platform comes with handles and wheels for easy displacement. It has one proprietary output, which connects to the scale indicator. Note that the platform itself does not have any power connector, it relies on the scale indicator for the power delivery. The four feet underneath are the sensor so make sure to handle them with care.

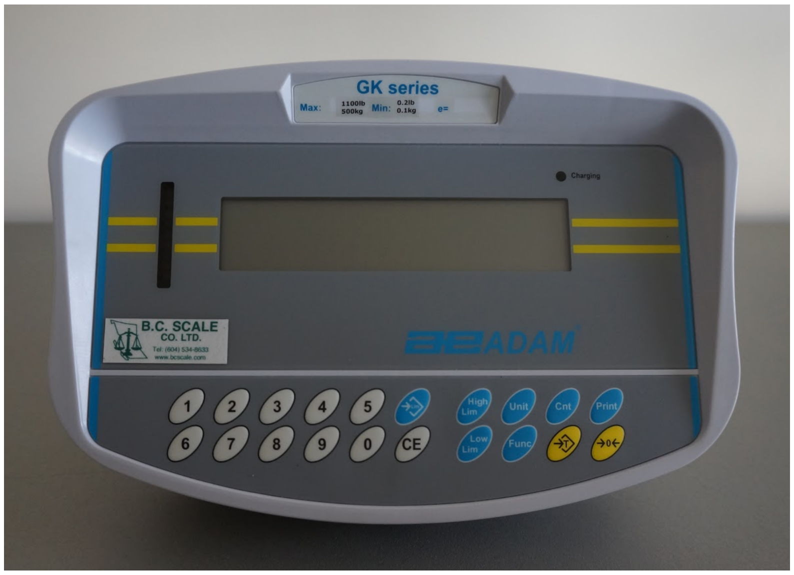
Specs:

33 x 31in (82 x 78cm)

74lb (33.5kg)

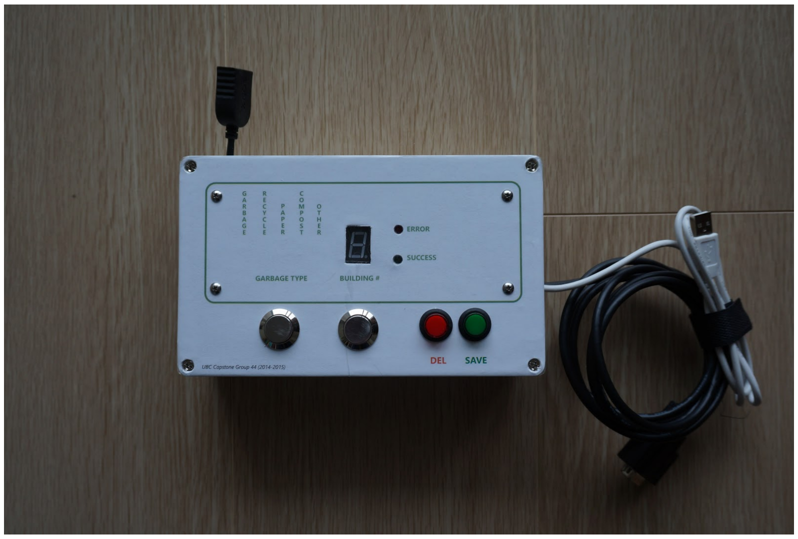
Link for more scale spec: http://www.adamequipment.com/docs/live/1204.pdf

Scale Indicator



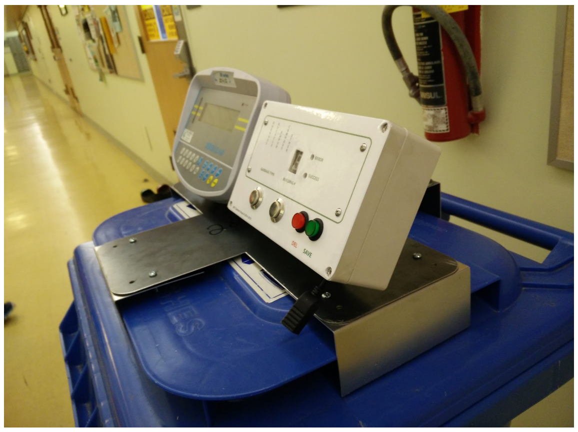
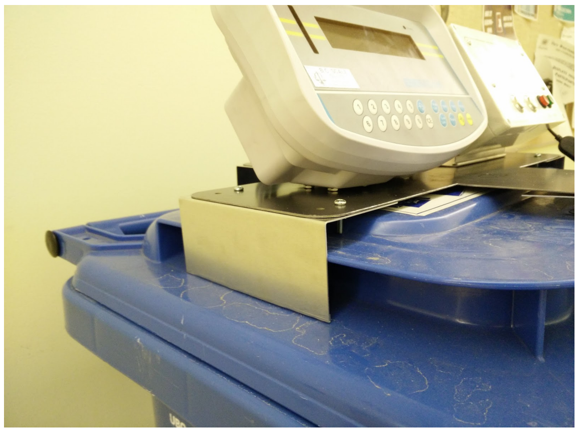
Description: The scale Indicator is responsible for displaying the weight on the scale platform. It communicates with the scale platform with a proprietary cable. Note that it tares the scale at the indicator's power on, so make sure the scale platform is clear. The indicator comes with a large battery and can be charged with the provided ac adapter

Controller



Description: The controller connects to the indicator with a RS-232 cable. It is responsible for interfacing with the user and output the data to a USB drive. The Raspberry Pi inside runs on 5v and can be powered via usb wall wart, we recommend powering it with a Power Bank to avoid needing the access to a wall outlet.

Mounting bracket

Description: The mounting bracket clamps on securely on the garbage bin, It eliminates the need of carrying a separate stand. The notches at the rear of the plate are for cable ties. The four 'claws' can be modified if ever the bin's dimension were to change in the future.

Setup

1. Lay the scale platform on a flat surface, make sure all four feets underneath are touching the ground.
2. Mount the interface board on a garbage bin, make sure it is secure.
3. Connect the scale indicator to the scale platform with the proprietary cable
4. Power ON the Scale Indicator
5. Power ON the controller
6. Make sure to tare an empty bin before recording on the scale indicator to correctly determine the net weight.

Operation

1. After power on, or after a 2 minutes timeout, the controller panel should have both the 5 LEDs "Garbage Type" and the 7-segment display labelled "Building Number" scrolling. All inputs are currently invalid and will will need further user input.

2. Select the desired “Garbage Type” and “Building Number”. (The green LED on the save button should be ON by now)
3. Place the bin on the scale.
4. Press “SAVE”.
5. Waiting until the “Success” LED turns on then remove the bin off the scale.
6. Repeat step 2-5 for all other bins

If the “Error” LED turns on, check the number displaying on the 7-seg display, then go to the troubleshooting section for the error details.

Delete previous entry

1. Press “DEL” after a mistake.
2. Success LED should light up for a few seconds after the entry has been successfully deleted.

Note that only one previous entry can be deleted.

USB Storage

The Raspberry Pi on the controller supports USB storage. To use collect the gathered data, plug in the USB storage device before powering it up. Then power it up as any other operations. The program will load the temporary data recorded on to the USB storage and any other data during operation. When USB storage is not used, the Raspberry Pi will record the data and store it on its local storage on the SD card. Next time a USB storage is found on the device, this data is copied on to the USB storage and delete off the local storage.

Charging

The scale indicator is charged with the provided AC adapter. The controller is power by a power band, with 2 powered USB port. The power bank is charged via its micro-usb port on the side. We recommend using a 2 amp, 5v USB wall wart (the ones for charging phones and tablets) for faster charging.

Troubleshooting

Error code 1:

If the 7-seg display is blinking a ‘1’, this indicates a reading timeout by controller to the scale indicator. Check the connection between the scale indicator and the controller. Other possible reasons for this error is also the scale indicator itself. It is best for the scale indicator to be weighing a steady weight.

Error code 2:

If the 7-seg display is blinking a ‘2’, this indicates that the weight read is invalid. Invalid weight values are under 1kg, which usually indicates that there is nothing on the scale. Check the scale indicator to correctly determine the weight.

Weight displayed incorrectly:

Press the tare button on the scale indicator

Controller unresponsive:

Check for connection, remove any weight from the scale platform then reboot both the scale indicator and the controller.

Raspberry Pi Access

To access the Raspberry Pi and configure its system, simply use an HDMI cable to connect to a HDMI display and use an USB keyboard and mouse to control its interface.

To access the Raspberry Pi via laptop, use an ethernet cable to connect from the laptop to the Raspberry Pi ethernet port. This is done using SSH to the Raspberry Pi. To set SSH up, remove the micro SD Card from the Raspberry Pi and put the SD card into a computer. In the boot folder, configure the cmdline.txt to support IP address for the laptop. For details, find the settings online. Also setup the network connections for ethernet. Look for it online as well. Use a SSH program to connect to the Raspberry Pi.

For connection via laptop instructions:
https://pihw.wordpress.com/guides/direct-network-connection/