University of Nebraska – Lincoln
Department of Psychology
Center for Brain, Biology, and Behavior
Dr. Jeffrey Stevens

Treat & Train Retrofit Documentation and User Guide
17 May 2020
V1.4

Overview

The *PetSafe Treat & Train Manners Minder Remote Reward Dog Trainer* is a commercial, at-home tool for performing operant conditioning with your dog. Using an RF remote control, the dispensation of treats can be controlled by the user based on the settings indicated by the switch interface on the top of the device.

Figure 1 – Treat & Train Device

A main operation of the device is to train your dog to perform a "Down/Stay" command, which is facilitated by the dispensation of a pre-determined number of treats.

Treat & Train Operation

When the Treat & Train is operated in the single treat dispensation mode, meaning that a single treat is dispensed when the 'Dispense' button on the RF remote is clicked, it monitors two devices:

1. IR break sensor on the treat dispensing eyelet,
2. IR break sensor on an internal mirror locked to the rotation of the dispensing wheel.

There are two stop conditions for the dispensing wheel:

1. The eyelet break sensor is triggered,
2. The wheel performs a full rotation.

These two conditions are the main driving signals for operation of the device.

Retrofitted Circuit Description

The functionality of the Treat & Train was analyzed, and reverse engineered and reimplemented on a custom board, which allows for future expansion and software-based upgrades. The IR sensors that are used for the primary operation of the device are hooked up to resistor divider bridges to act as a voltmeter. The on-board motor driver interfaces to the main DC motor inside of the device. The motor output of the Treat & Train is hooked up to a transistor to detect when the main board is requesting a treat. The system voltage is boosted to 12V to drive the DC motor and the entire system operates off of the USB power input.

Prototype Board

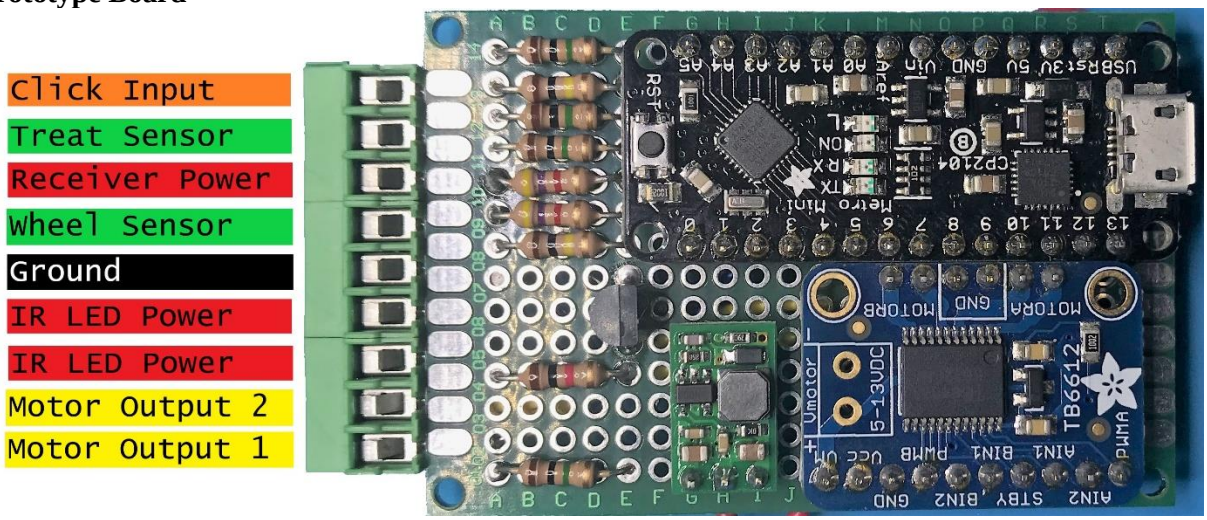


Figure 2 – Working Circuit on Protoboard including Pinout

Pin Name	Arduino Pin	Function	Description
Motor Output 2	N/A	Output	Connected to negative motor pin
Motor Output 1	N/A	Output	Connected to positive motor pin
IR LED Power	N/A	Output	5V power supply for IR LEDs, current limited to 5mA
Ground	N/A	Input	Common ground pin
Wheel Sensor	A1	Input	IR break sensor output from wheel sensor
Receiver Power	N/A	Output	5V power supply for break sensors, current limited to 1mA
Treat Sensor	A3	Input	IR break sensor from treat sensor
Click Input	4	Input	RF remote input for ‘Dispense’ button

The on-board motor driver is connected to the Adafruit Metro Mini, which is an equivalent design to the Arduino Uno. The outputs from the Metro Mini are defined below.

Pin Name	Arduino Pin	Description
BIN1	9	First input for motor driver, invert from other input to drive motor in one direction or the other.
BIN2	8	Second input for motor driver, invert from other input to drive motor in one direction or the other.
PWMB	6	PWM input that controls the speed of the motor, accepts values between 0-255. Default: 127

At this time, there is no working implementation using the wheel sensor.

When using the RF remote to trigger the system, **the Treat & Train main board must first be plugged in before plugging in the retrofitted board.**

Do not use anything other than a computer USB port for powering the system. The retrofitted board does not power the Treat & Train main board and the main device will still require four D cell batteries to operate.

Custom Printed Circuit Board
The custom board implements the functionality of the prior prototype board, with a fully integrated hardware solution.

Figure 3 – Custom Printed Circuit Board

Pin Name	Arduino Pin	Function	Description
M+	N/A	Output	Connected to positive motor pin
M-	N/A	Output	Connected to negative motor pin
GND	N/A	Input	Common ground pin
IR1	N/A	Output	3.3V power supply for IR LEDs, current limited to 5mA
IR2	N/A	Output	3.3V power supply for IR LEDs, current limited to 5mA
TRT	A1	Input	IR break sensor from treat sensor
BRK	N/A	Output	3.3V power supply for break sensors, current limited to 1mA
WHL	A3	Input	IR break sensor output from wheel sensor
CLK	D8	Input	RF remote input for ‘Dispense’ button

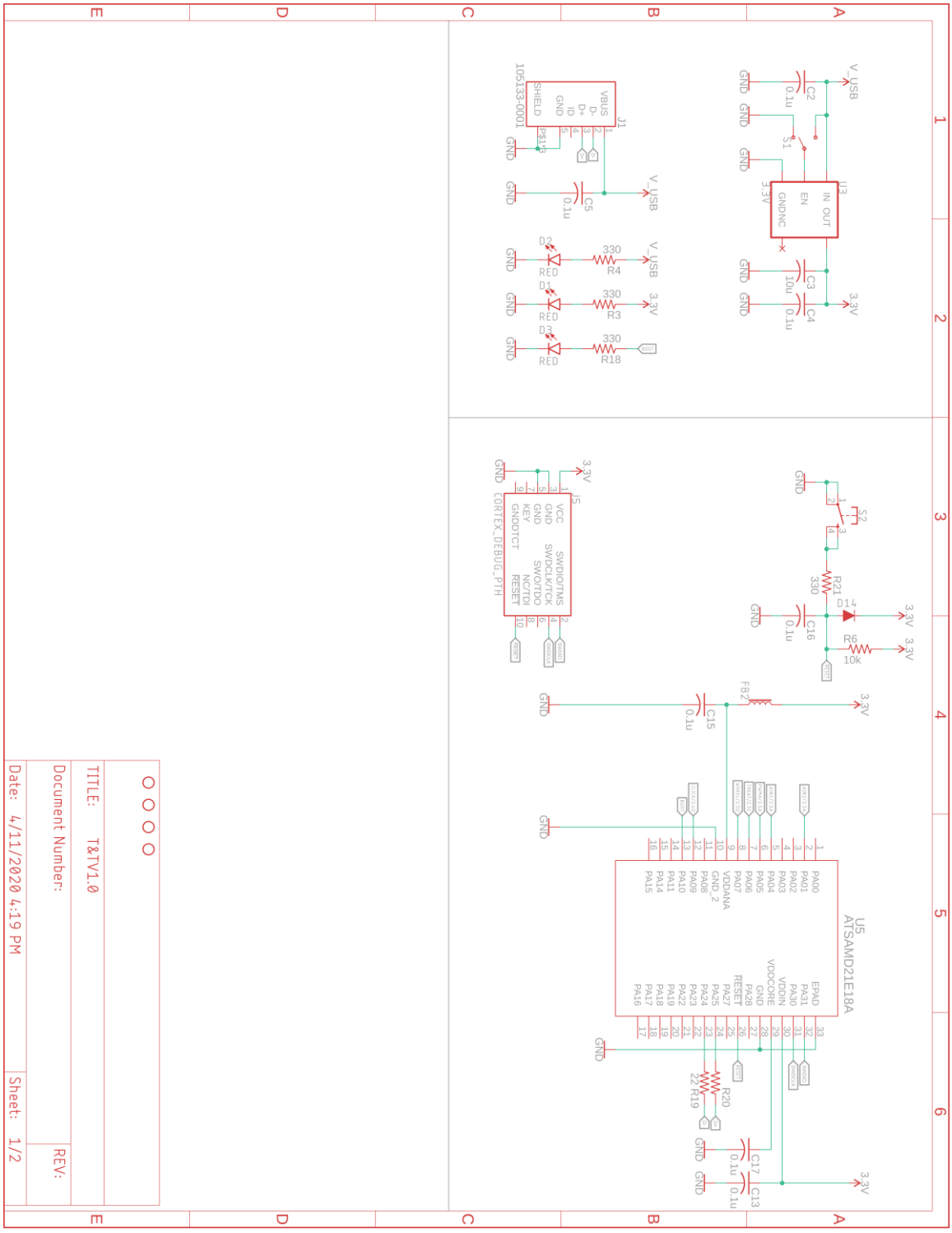
Pin Name	Arduino Pin	Description
AIN1	D7	First input for motor driver, invert from other input to drive motor in one direction or the other.
AIN2	D6	Second input for motor driver, invert from other input to drive motor in one direction or the other.
PWMA	D3	PWM input that controls the speed of the motor, accepts values between 0-255. Default: 127

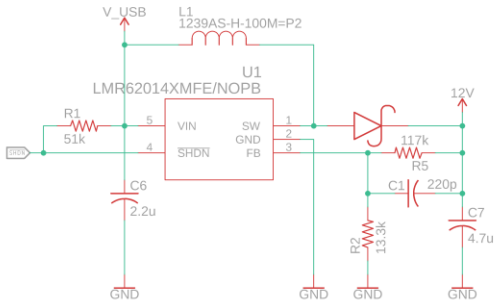
At this time, there is no working implementation using the wheel sensor.

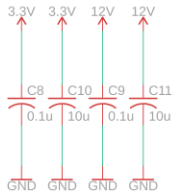
When using the RF remote to trigger the system, **the Treat & Train main board must first be plugged in before plugging in the retrofitted board.**

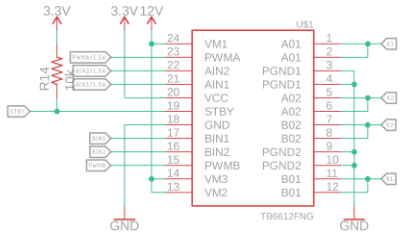
Do not use anything other than a computer USB port for powering the system. The retrofitted board does not power the Treat & Train main board and the main device will still require four D cell batteries to operate. **This board shows up as the “Sparkfun Qwiic Micro” when seen in the Arduino IDE.**

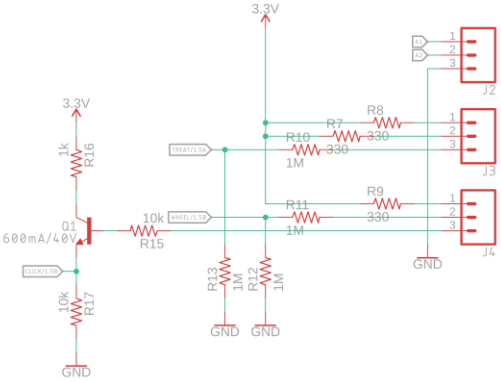
Schematic


TITLE: T&TV1.0
Document Number:
REV:
Date: 4/11/2020 4:19 PM
Sheet: 1/2
U5
ATSAMD21E18A
J1
105133-0001
J5
CORTEX_DEBUG_PTH
U3
S1
S2
D1
D2
D3
D4
R3
R4
R6
R18
R19
R20
R21
C2
C3
C4
C5
C13
C15
C16
C17
FB2
V_USB
3.3V
GND
RESET
SWDIO
SWCLK









TITLE: T&TV1.0		
Document Number:		REV:
Date: 4/11/2020 4:19 PM	Sheet: 2/2	

USB Protocol

The system operates using a defined USB protocol driven by byte codes. This can be easily expanded upon in the firmware by adding cases to the switch statement for the command byte. This is defined below.

Name	**Value**	**Function**
Start Byte	67 (Dec), 0x43, 'C'	Signals the system to start listening for input
Command Byte	1 byte	Command defined below
Data Byte	1 byte	Defined based on commands
End Byte	69 (Dec), 0x45, 'E'	Signals the system to stop listening for input

There are a few pre-defined commands for this version of the firmware, one of which is not accessible through USB input.

Command	**Byte**	**Data**	**Function**
Dispense	'D'	0-9 (ASCII)	Command to dispense treats, the number of treats is between 0 and 9 in ASCII.
Print	'P'	0 (ASCII)	Prints current value of sensors, data field is set to zero in ASCII
Update RF	'B'	0-9 (ASCII)	Updates the number of treats dispensed when using the RF remote.
Dispense	'A'	0-255 (ASCII)	Command to dispense treats, the number of treats is interpreted as the hex value of the ASCII char for data
Motor Update	'M'	0-255 (ASCII)	Updates the motor speed used when operating the dispenser in any mode
Wheel Test	'F'	0 (ASCII)	Performs the wheel test routine to verify operation of the device and sensors
RF Dispense	N/A	N/A	By clicking the 'Dispense' button on the RF remote, the system will dispense the defined number of treats. Default: 1 treat

Python Class Definition

```
class CanineDispenser
	__init__(comport_name='/dev/ttyACM0', delay=1)
		Parameters:  comport_name – device name, defaults to Linux device name
		             delay – period in seconds to delay to account for reset
	update_rf_amount(num_treats)
		Parameters:  num_treats – the number of treats to dispense, int [0, 9]
		Returns: 0 for success, 1 for failure, byte
	dispense_treats(num_treats)
		Parameters:  num_treats – the number of treats to dispense, int [0, 9]
		Returns: 0 for success, 1 for failure, byte
	dispense_treats_multi(num_treats)
		Parameters:  num_treats – the number of treats to dispense, int [0, 255]
		Returns: 0 for success, 1 for failure, byte
	change_motor_speed(motor_speed)
		Parameters:  motor_speed – new motor speed, int [0, 255]
		Returns: 0 for success, 1 for failure, byte
	debug_sensors()
		Returns: output line of the dispenser for state of sensors, str
	wheel_test()
		Returns: None. Outputs sensor information to command line
	disconnect()

list_comports()
	Returns: list containing serial objects that are available, list
read_dispenser_line(dispenser)
	Parameters:  dispenser – CanineDispenser object to operate on
	Returns: string containing dispenser output, str
dispenser_test(expected_dispensers)
	Parameters:  expected_dispensers – number of dispensers to connect to, int
	Returns: None. Outputs information to the command line
dispenser_functionality_test(comport_name)
	Parameters:  comport_name – string representing COM port to connect to
	Returns: None. Outputs information to the command line
```

Interfacing Example

The retrofitted board can be interfaced using any modern system, but for this document Python 3.7 is used to interface to the system. A single library is needed, pySerial, which can be installed using Pip.

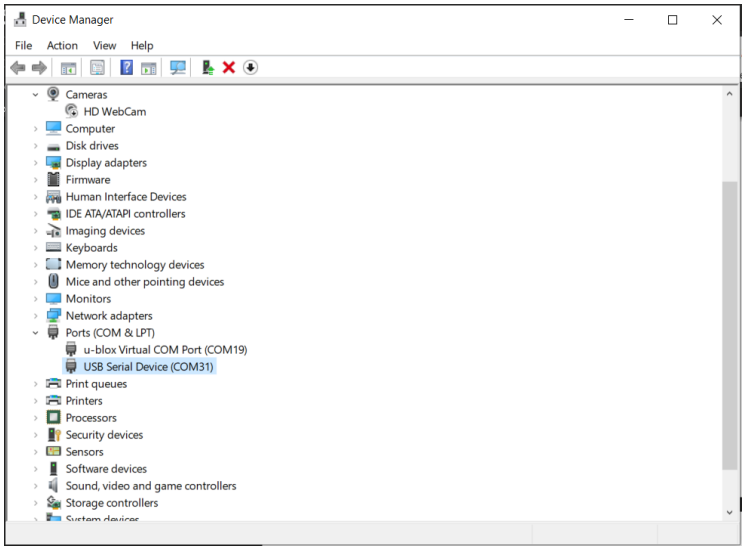
```
comports = list_comports()

dispensers = []

while expected_dispensers != 0:
    try:
        com_select = input("Select COM port from list above: ")
        if int(com_select) < 0 or int(com_select) > len(comports) - 1:
            print('Invalid input. Valid range is 0 to', len(comports) - 1)
        else:
            dispensers.append(CanineDispenser(comport_name=comports[int(com_select)].device))
            expected_dispensers -= 1
    except:
        print('Invalid COM port selected, use numerals in range [0, 9]. Returning control.')
        return

while True:
    command = input('Enter number of treats to dispense [0, 9] or "exit" to quit: ')
    try:
        if command is 'exit':
            for dispenser in dispensers:
                dispenser.disconnect()
            return
        elif int(command) < 0 or int(command) > 9:
            print('Invalid input. Valid range is [0, 9].')
        else:
            for dispenser in dispensers:
                out = dispenser.dispense_treats(command)
                if out == b'1':
                    print('Invalid command for', dispenser.comport.name)
                elif out == b'0':
                    print('Successful command for', dispenser.comport.name)
    except:
        print('Invalid input. Only use numerals [0, 9] or type "exit". Returning control.')
        return
```

Notice below that the new Treat & Train shows up as *USB Serial Device* on COM31.



Raspberry Pi 4 as an Experiment Platform

The experiments on canine operant conditioning are run using a Python package called PsychoPy. It essentially acts as a development environment for psychology experiments and provides a variety of utilities for user interface development, trigger conditions, experiment logging, and so on. Our aim is to integrate this directly into a Raspberry Pi 4, to allow the entire setup to become a mobile testing unit. To do this, we will have a few requirements:

1. Wireless Access Point
    a. The Raspberry Pi 4 should host its own WiFi access point (AP), which allows the experimenter to connect to the Pi and run experiments, download results, and interact with the main system.
2. Decoupled Interfaces
    a. The screen the experimenter will use, and the screen used by the canine will be separate and require no extra hardware to achieve. By connecting to the Pi using SSH (Secure Shell), the command line can be exposed to run the experiments. The canine screen can be connected to the HDMI0 of the Pi and the SSH session can be started in a separate area.
3. Simple Setup
    a. We want to make sure this is easy to setup and maintain. This means that we would want to use robust packages such as 'pi-ap' and SD card backups to easily restore the device and get new devices up and running easily.

There are two ways to setup a new Raspberry Pi to operate the canine dispenser:

1. Flash a new SD card with an image from the iso_images folder
    a. Using a Raspberry Pi with a fresh installation of Raspbian Buster, there are built-in tools to achieve this. "The SD Card Copier application, which can be found on the Accessories menu of the Raspberry Pi Desktop, will copy Raspbian from one card to another. To use it, you will need a USB SD card writer. To back up your existing Raspbian installation, put a blank SD card in your USB card writer and plug it into your Pi, and then launch SD Card Copier. In the 'Copy From Device' box, select the internal SD Card. This could have a number of different names and may have something like (/dev/mmcblk0) in its entry, but will usually be the first item in the list. Then select the USB card writer in the 'Copy To Device' box (where it will probably be the only device listed). Press 'Start'. The copy, depending on the size of the SD card, can take ten or fifteen minutes, and when complete you should have a clone of your current installation on the new SD card. You can test it by putting the newly-copied card into the Pi's SD card slot and booting it; it should boot and look exactly the same as your original installation, with all your data and applications intact."
2. Follow the proceeding instructions to setup the Raspberry Pi manually.
    a. Flash a new copy of Rasbian Buster onto a micro SD card and place an empty file named "ssh" onto the boot partition. There is no file extension.
    b. Connect the Raspberry Pi to your local network using an Ethernet cable and power up the Pi. You can connect a monitor, keyboard, and mouse and use `ifconfig` to find the IP address of the Pi.
    c. Using this IP address, go to PuTTY and type in this IP address and connect to the Pi. The default password for "pi" is "raspberry".
    d. Once in, the pi-ap software can be installed to create the access point. Note, the dog_operant repository does contain a copy of this, but the main repository will contain the most recent version.
    e. The dog_operant repository can be cloned using the `git clone` command.
    f. Run the command `sudo apt-get update`

g. You will need to upgrade pip using
```
python3 -m pip install --upgrade pip
```
which will allow it to index some of the following packages.

h. Run the command
```
python3 -m pip install psychopy==3.2.4
```
which will install all the needed packages. Using the older version of PsychoPy will prevent a 'Segmentation fault' caused when rendering a new Window.

i. You may get an error saying that there is no package called 'wx', in which case you need to run the command
```
python3 -m pip install wxPython
```

j. If you want to run experiments, you will need to connect a screen to the Pi on HDMI0. An issue may present itself where no screen is presented when connected, which is caused by an incorrect /boot/config.txt file. To fix this do the following:
    i. `sudo nano /boot/config.txt`
    ii. Add the following to the file or uncomment from the file,
        1. `hdmi_force_hotplug=1`
        2. `hdmi_drive=2`
    iii. If this does not fix the issue, add,
        1. `hdmi_safe=1`

k. To run an experiment, use the following command
```
export DISPLAY=:0 ; python3 dog_operant/system_test.py
```
Note, this works from the command line, so run this when you are connected to an SSH session and with a screen connected to HDMI0 of the Pi and the experiment will display over HDMI0. Setting the DISPLAY variable to 0 will default it to the local display.

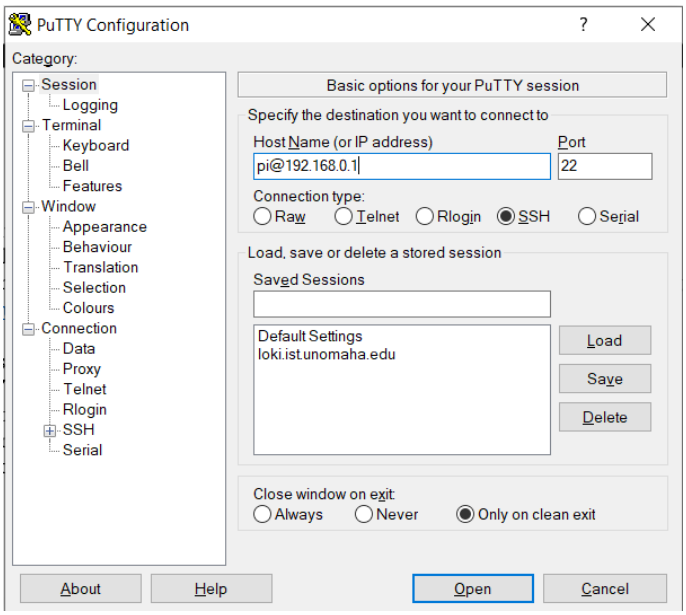
3. To transfer the results of the experiments, copy the contents of the data directory from the dog_operant directory. Use the following command structure:

```
scp -r user@ssh.example.com:/path/to/remote/source /path/to/local/destination
```

Raspberry Pi Interfacing and Passwords

The dispenser, when turned on, will create a WiFi access point called, “operant-canine-xx”, with “xx” being the number, i.e. “01”. By connecting to this, the Raspberry Pi can be accessed through SSH using a program called PuTTY.

Local Address	Password
`pi@192.168.0.1`	`cb3stevens`
`pi@adml-stevens-01`	`cb3stevens`



Bill of Materials

Printed Circuit Board Parts

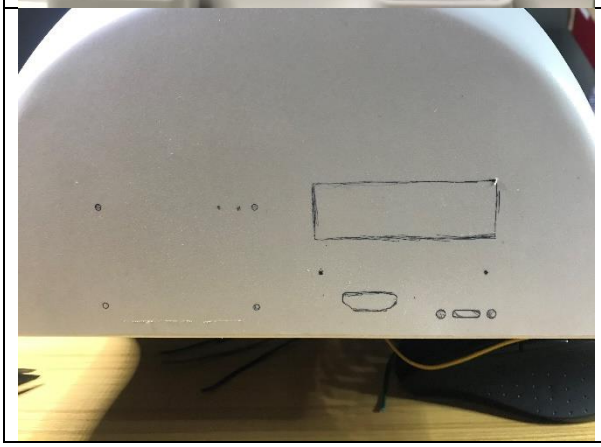
Bill of Materials - Treat&Train Retrofit Board						
Part	Description	Supplier	Part Number	Quantity	Price	Final Price
ATSAMD21E	Microcontroller	DigiKey	ATSAMD21E18A-MUTCT-ND	1	$2.92	$ 2.92
TB6612FNG	Motor Driver	DigiKey	TB6612FNGC8ELCT-ND	1	$1.95	$ 1.95
LMR6201	12V Boost	DigiKey	LMR62014XMFE/NOPBCT-ND	1	$1.64	$ 1.64
AP2112K	3.3V LDO	DigiKey	AP2112K-3.3TRG1DICT-ND	1	$0.49	$ 0.49
Vertical USB	USB Connector	DigiKey	WM9734CT-ND	1	$1.11	$ 1.11
Screw Terminal	Sensor Connectors	DigiKey	ED10562-ND	3	$1.03	$ 3.09
Debug Header	Programming Header	DigiKey	609-3712-ND	1	$0.69	$ 0.69
SPST Switch	Reset Button	DigiKey	CKN12223-1-ND	1	$0.10	$ 0.10
SPDT Switch	Power Switch	DigiKey	401-2016-6-ND	1	$0.91	$ 0.91
Stand Off	Board Mounting	DigiKey	36-24434-ND	4	$0.42	$ 1.68
Ferrite Bead	Analog Supply	DigiKey	445-172886-1-ND	1	$0.10	$ 0.10
MMBT2222AL	Click NPN Trans	DigiKey	MMBT2222ALT1GOSCT-ND	1	$0.13	$ 0.13
BAS16J	Weather Sensor Diodes	DigiKey	1727-6169-1-ND	1	$0.18	$ 0.18
LED 0603	Indicator LEDs	DigiKey	511-1588-1-ND	3	$0.44	$ 1.32
MBR0520LT1	12V Diode	DigiKey	MBR0520LT1GOSCT-ND	1	$0.29	$ 0.29
10u Ind	12V Inductor	DigiKey	490-10563-1-ND	1	$0.42	$ 0.42
220p 0603 Cap	12V Stability Cap	DigiKey	311-4111-1-ND	1	$0.10	$ 0.10
0.1u Cap	Bypass Caps	DigiKey	399-1100-1-ND	9	$0.10	$ 0.90
10u Cap	LF Filtering	DigiKey	311-2007-1-ND	3	$0.23	$ 0.69
2.2u 0603 Cap	12V Filtering	DigiKey	399-7886-1-ND	1	$0.14	$ 0.14
4.7u 0603 Cap	12V Filtering	DigiKey	399-3482-1-ND	1	$0.19	$ 0.19
117k 0603 Res	12V Bias Res	Mouser	603-RT0603DRE07117KL	1	$0.13	$ 0.13
51k 0603 Res	12V SHDN Pullup	DigiKey	749-1754-1-ND	1	$0.17	$ 0.17
13.3k 0603 Res	12V Bias Res	DigiKey	RMCF0603FT13K3CT-ND	1	$0.10	$ 0.10
22 Res	USB Resistors	Digikey	A129668CT-ND	2	$0.10	$ 0.20
330 Res	LED Resistor	DigiKey	P330HCT-ND	7	$0.10	$ 0.70
10k Res	Pull Up Res	DigiKey	311-10KGRCT-ND	4	$0.10	$ 0.40
1k 0603 Res	Click Current Res	DigiKey	311-1.0KGRCT-ND	1	$0.10	$ 0.10
1M 0603 Res	Sensor Divider Res	DigiKey	311-1.00MHRCT-ND	4	$0.10	$ 0.40

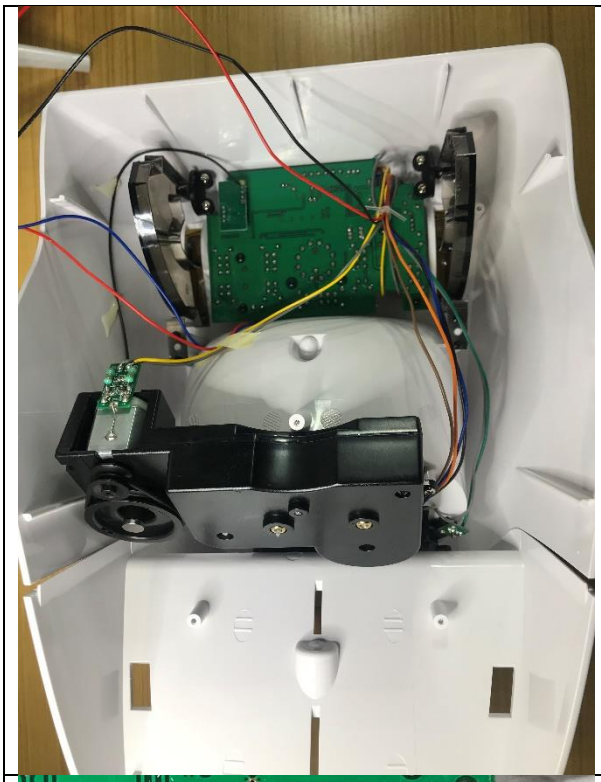
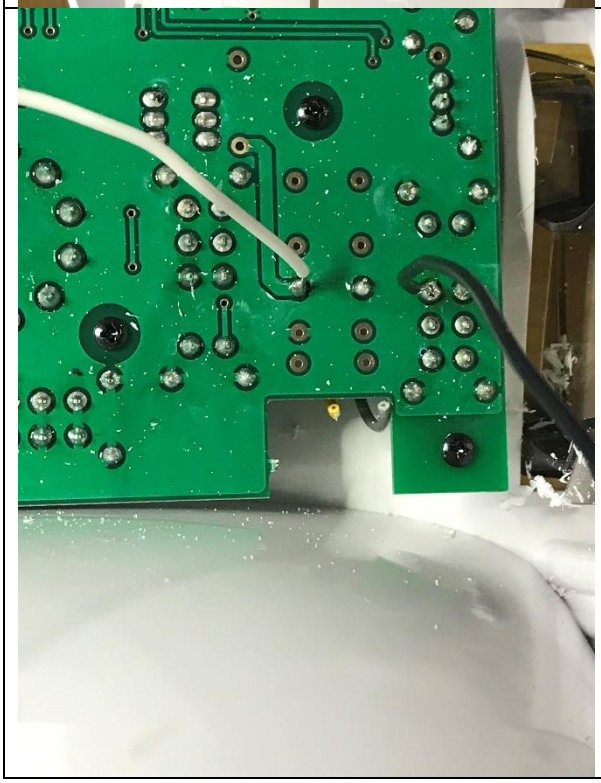
Build Parts

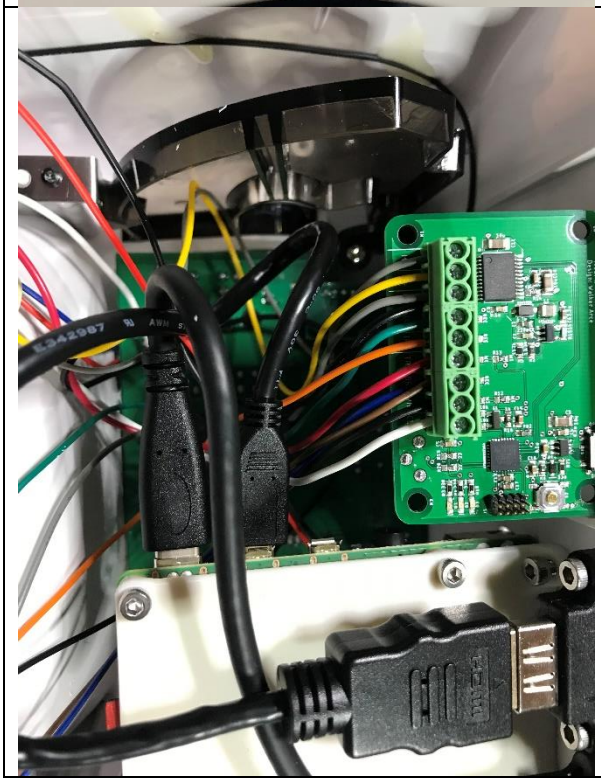
Part	Supplier	Quantity	Price
Raspberry Pi 4	Sparkfun	1	$ 59.95
16 GB SD Card	Amazon	1	$ 6.99
Power Supply	Sparkfun	1	$ 8.00
USB C Panel Mount Cord	Amazon	1	$ 8.99
HDMI Panel Mount	Amazon	1	$ 1.59
Micro HDMI to HDMI Cord	Amazon	1	$ 5.41
Micro USB to USB A Cord	Sparkfun	1	$ 1.95

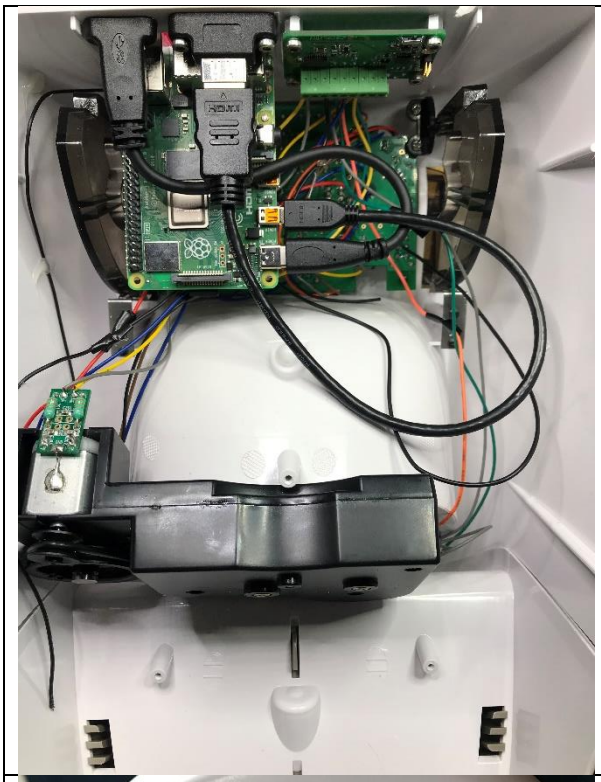
Build Instructions

	1. Flip the Treat & Train over and remove the five screws holding the bottom plate on.
	2. Next, flip the device over, open the treat reservoir and locate the two screws on the wheel face and remove them. Remove both this plate and the bottom plate, flipping over the device to expose the internal electronics.

	3. Observe the front exposed face and locate the black wire connected to the top IR receiver. This is marked in red. Cut it and solder on a new red wire approximately four inches in length. This is the output signal labeled ‘TRT’ on the custom board.
	4. Flip the device around and mark on the back face where the new electronics will be exposed.

	5. Now, observe the internal electronics of the stock device and notice the cord bundle on the top right of the device. Cut all of these except for the red and black power wires that go to the battery bank. Strip the ends of the cut wires and place them to the side.
	6. Next, tack on two wires to the shown pads on the underside of the internal board. The white wire will be our 'Click' signal and the black will be the common ground of the stock electronics.

	7. Bore out the marked ports on the back of the device and install the electronics using M3 screws. Eight will be required of the 12mm variety. Additionally, notice the teeth on the inner wall of the bottom plate. Clip these off using flush cutters as the installed electronics will get in the way of them.
	8. Now, install the wires from the internals of the device to the custom board shown.

	9. Note, for this build we used as custom a Raspberry Pi, right angle mounting plate. Now, connect the associated wires to Pi.
	10. Screw the device back together and plug in the short USB cable from the Pi to the custom board.

Firmware

pins.h

```
#define MDRIVERA 7
#define MDRIVERB 6
#define MSPEED 3
#define CLICK 8
#define WHEEL_SENSOR A3
#define TREAT_SENSOR A1
```

TreatAndTrainDriverCustomBoard.ino

```
/*
 * Treat & Train Custom Driver
 * Author: Walker Arce, May 2020
 *
 * This program is used to drive the Treat & Train device for pellet dispensation.
 * It operates using a simple USB command protocol and can also be driven by the use of the
 * accompanying RF remote.
 *
 * The current operation of the system is as follows:
 *    1. A command is sent over USB to dispense a pellet,
 *        a. Drives wheel until pellet sensor is broken or wheel goes a full rotation.
 *    2. The button 'Dispense' is clicked on the RF remote
 *        a. Drives wheel until pellet sensor is broken or wheel goes a full rotation.
 *
 * The motor can be driven at a variety of levels, for the second state of operation, it will be driven at half
speed (6V DC).
 *
 * This initial version is meant to simply replicate functionality.  The Treat & Train can be restored to
original functionality
 * by reconnecting the internal board and sensors.
 *
 * v0.3
 * 12 May 2020
 */

#include "pins.h"

#define Serial        SerialUSB
#define SerialDebug   false
#define VOLTAGE_SCALE 3.3
#define ADC_PRECISION 4096.0
#define LOGIC_LOW     0.09

//Voltage sensing variables
float Vout = 0.00;
float Vin = 0.00;
float R1 = 1000000.00; // resistance of R1 (1M)
float R2 = 1000000.00; // resistance of R2 (1M)

//Peripheral sampling variables
float sampleVoltage = 0;
float wheelVoltage = 0;
float treatVoltage = 0;
int clickSignal = 0;

int clickCycles = 1;
int globalMotorSpeed = 127;

void setup()
{
   Serial.begin(9600);
   analogReadResolution(12); // Set analog input resolution to max, 12-bits
   initPins();
   stopMotor();
   //while(1){ debugSensors(); }
   //moveMotor(125, true);
}

void loop()
{
  //Listen for the RF remote
  if (digitalRead(CLICK) == HIGH)
  {
    if(dispenseCommand(clickCycles, 127, false, false))
    {
      Serial.println("Successfully dispensed");
    }
```

```
    else
    {
      Serial.println("Error occurred");
    }
    while(digitalRead(CLICK) == HIGH) {}
  }
  //Wait for USB input
  while(Serial.available() > 0)
  {
    char command[4];
    size_t bytesRead = Serial.readBytes(command, 4);
    //Check for 'C' being sent
    if (command[0] == 67 && command[3] == 69)
    {
      switch(command[1])
      {
        //Check for 'D' being sent
        //This dispenses the specified number of treats, give or take.
        case 68:
          if (dispenseCommand(command[2], globalMotorSpeed, false, true))
          {
            Serial.print(0x00, HEX);
          }
          else
          {
            Serial.print(0x01, HEX);
          }
          break;
        //Check for 'P' being sent
        //This prints out the current state of the internal sensors.
        case 80:
          debugSensors();
          break;
        //Check for 'B' being sent
        //This updates the number of treats dispensed when RF remote is clicked
        case 66:
          clickCycles = command[2] - 0x30;
          Serial.print(0x00, HEX);
          break;
        //Check for 'A' being sent
        //This is meant to be used when more than nine treats are expected to be dispensed
        //Does not convert ASCII character to decimal
        case 65:
          if (dispenseCommand(command[2], globalMotorSpeed, false, false))
          {
            Serial.print(0x00, HEX);
          }
          else
          {
            Serial.print(0x01, HEX);
          }
          break;
        //Check for 'M' being sent
        //This allows the changing of the global motor speed
        //Not recommended to be used
        case 77:
          globalMotorSpeed = command[2];
          Serial.print(0x00, HEX);
          break;
        //Command not recognized
        default:
          Serial.print(0x01, HEX);
      }
    }
    else
    {
      Serial.print(0x01, HEX);
    }
  }
}

void testWheel()
{
  int motorSpeed = 0;
  while(motorSpeed < globalMotorSpeed)
  {
    moveMotor(motorSpeed++, true);
    wheelVoltage = getVoltage(WHEEL_SENSOR, 10);
    wheelVoltage = roundFloat(wheelVoltage, 1);

    treatVoltage = getVoltage(TREAT_SENSOR, 10);
```

```
    treatVoltage = roundFloat(treatVoltage, 1);

    Serial.print("Wheel Voltage: ");
    Serial.print(wheelVoltage);
    Serial.print("V | Treat Voltage: ");
    Serial.print(treatVoltage);
    Serial.print("V");
    Serial.print(" | Speed: ");
    Serial.print(motorSpeed);
    Serial.println();
    delay(200);
  }
  stopMotor();
}

void debugSensors()
{
  moveMotor(127, true);

  wheelVoltage = getVoltage(WHEEL_SENSOR, 10);
  wheelVoltage = roundFloat(wheelVoltage, 1);

  treatVoltage = getVoltage(TREAT_SENSOR, 10);
  treatVoltage = roundFloat(treatVoltage, 1);

  Serial.print("Wheel Voltage: ");
  Serial.print(wheelVoltage);
  Serial.print("V | Treat Voltage: ");
  Serial.print(treatVoltage);
  Serial.print("V");
  Serial.println();
  delay(200);
}

bool dispenseCommand(int cycles, int motorSpeed, bool motorDir, bool isASCII)
{
  sampleVoltage = getVoltage(TREAT_SENSOR, 10);
  sampleVoltage = roundFloat(sampleVoltage, 1);

  int treatSamples = cycles;
  if (isASCII) { treatSamples = cycles - 0x30; }
  int wheelSamples = treatSamples * 3;
  moveMotor(motorSpeed, motorDir);

  while (treatSamples > 0)
  {
    treatVoltage = getVoltage(TREAT_SENSOR, 10);
    treatVoltage = roundFloat(treatVoltage, 1);

    if (treatVoltage < sampleVoltage)
    {
      treatSamples--;
      while (treatVoltage < sampleVoltage)
      {
        treatVoltage = getVoltage(TREAT_SENSOR, 10);
        treatVoltage = roundFloat(treatVoltage, 1);
      }
    }
  }
  stopMotor();
  return true;
}

//Source: https://stackoverflow.com/questions/1343890/how-do-i-restrict-a-float-value-to-only-two-places-after-
the-decimal-point-in-c
float roundFloat(float raw, int precision)
{
  return floorf(raw * (10 * precision)) / 10;
}
//Source: https://create.arduino.cc/projecthub/next-tech-lab/voltmeter-using-arduino-00e7d1
float getVoltage(int pin, int samples)
{
  int val = getAverageReading(pin, samples);
  Vout = (val * VOLTAGE_SCALE) / ADC_PRECISION; // formula for calculating voltage out i.e. V+, here 5.00
  Vin = Vout / (R2/(R1+R2)); // formula for calculating voltage in i.e. GND
  if (Vin < LOGIC_LOW)//condition
  {
   Vin=0.00;
  }
  return Vin;
}
```

```
int getAverageReading(int pin, int numReadings)
{
  int adcReading = 0;
  for (int i = 0; i < numReadings; i++)
  {
    adcReading += analogRead(pin);
  }
  adcReading /= numReadings;
  return adcReading;
}

void moveMotor(int motorSpeed, bool motorDir)
{
  //Set motor speed before driving motor
  analogWrite(MSPEED, motorSpeed);
  //If true, then drive it clockwise
  if (motorDir)
  {
    digitalWrite(MDRIVERA, HIGH);
    digitalWrite(MDRIVERB, LOW);
  }
  //If false, then drive it counter-clockwise
  else
  {
    digitalWrite(MDRIVERA, LOW);
    digitalWrite(MDRIVERB, HIGH);
  }
}

void stopMotor()
{
  analogWrite(MSPEED, 0);
  digitalWrite(MDRIVERA, LOW);
  digitalWrite(MDRIVERB, LOW);
}

void initPins()
{
  //Initialize inputs
  pinMode(WHEEL_SENSOR, INPUT);
  pinMode(TREAT_SENSOR, INPUT);
  pinMode(CLICK, INPUT);

  //Initialize outputs
  pinMode(MDRIVERA, OUTPUT);
  pinMode(MDRIVERB, OUTPUT);
  pinMode(MSPEED, OUTPUT);
}
```

dispenser_utilities.py

```
"""dispenser_utilities.py

This script creates a serial interface to the retrofitted Treat & Train device developed
for Dr. Jeffrey Stevens' canine operant conditioning studies.  The member methods
comply with the USB protocol developed to interact with its internal state machine
and encapsulates a comport definition and a configurable delay period.

Supported device operations include (replace X with relevant value):
    Dispense treats in [0, 9] range:                          CDXE
    Dispense treats in [0, 255] range:                        CAXE
    Update the RF remote dispensation amount in range [0, 9]:   CBXE
    Update motor speed in range [0, 255]:                     CMXE
    Debug sensors:                                            CP0E
    Wheel test routine:                                       CF0E

    Note: The value placed in X will be represented in ASCII, so for the 'D' command it will be valid in ASCII
range
    [0, 9], in which case the device will convert the value passed from ASCII to decimal.  In the case of the
'A'
    command, it treats the ASCII value as the number of treats and does not convert the value to decimal.

    Supports version 0.4 of the dispenser firmware.

Author:     Walker Arce
Date:       05/17/2020
Version:    0.3
"""

import serial
```

```
import serial.tools.list_ports
import time


class CanineDispenser:
    """A simple class definition to interact with the canine treat dispenser.

    self.comport - pyserial USB port object used to communicate with the device.
    self.delay - period in seconds to wait before sending command to allow for reset.
    """
    def __init__(self, comport_name='/dev/ttyACM0', delay=1):
        """
        Parameters
        :param comport_name: str
            The name of the USB port to connect to, i.e. '/dev/ttyACM0' or 'COM31'
        :param delay: int or float
            The period in seconds to wait before sending command to allow for reset
        """
        try:
            self.comport = serial.Serial(comport_name, baudrate=9600, timeout=None)
            self.delay = delay
        except serial.SerialException:
            print('Failed to open serial port, double check port and try again.')
            print('Port: ' + comport_name)

    def update_rf_amount(self, num_treats):
        """Sends a command to the dispenser to update the number of treats dispensed when using the RF remote.
        Parameters
        :param num_treats: int
            The number of treats in range [0, 9] to dispense
        :return: byte
            0 for success, 1 for failure
        """
        if num_treats < 0 or num_treats > 9:
            print('Number of treats must be within range [0, 9], please retry.')
            return
        if self.comport.isOpen():
            command = 'CB'.encode('ascii') + str(num_treats).encode('ascii') + 'E'.encode('ascii')
            time.sleep(self.delay)
            self.comport.write(command)
        return self.comport.read(1)

    def dispense_treats(self, num_treats):
        """Sends command to dispense treats in range [0, 9]
        Parameters
        :param num_treats: int
            The number of treats in range [0, 9] to dispense
        :return: byte
            0 for success, 1 for failure
        """
        if int(num_treats) < 0 or int(num_treats) > 9:
            print('Number of treats must be within range [0, 9], please retry.  '
                  'Use dispense_treats_multi to dispense more treats.')
            return
        if self.comport.isOpen():
            command = 'CD'.encode('ascii') + str(num_treats).encode('ascii') + 'E'.encode('ascii')
            time.sleep(self.delay)
            self.comport.write(command)
        else:
            print('No open comport on this object.')
            return
        return self.comport.read(1)

    def dispense_treats_multi(self, num_treats):
        """Sends command to dispense a greater amount of treats, in range [0, 255]
        Parameters
        :param num_treats: int
            The number of treats to dispense in range [0, 255]
        :return: byte
            0 for success, 1 for failure
        """
        if int(num_treats) < 0 or int(num_treats) > 255:
            print('Number of treats must be within range [0, 255], please retry.')
            return
        if self.comport.isOpen():
            command = 'CA'.encode('ascii') + chr(num_treats).encode('ascii') + 'E'.encode('ascii')
            time.sleep(self.delay)
            self.comport.write(command)
        return self.comport.read(1)

    def change_motor_speed(self, motor_speed):
```

```
        """Sends command to update the motor speed with provided value in range [0, 255]
        Note: Increasing the motor speed beyond 127 (default) is not recommended for expected use case.
        Parameters
        :param motor_speed: int
            Speed of the motor in range [0, 255] or [0V, 12V] equivalent
        :return: byte
            0 for success, 1 for failure
        """
        if motor_speed < 0 or motor_speed > 255:
            print('Motor speed must be within valid range [0, 255], please retry.')
            return
        if self.comport.isOpen():
            command = 'CM'.encode('ascii') + chr(motor_speed).encode('ascii') + 'E'.encode('ascii')
            time.sleep(self.delay)
            self.comport.write(command)
        return self.comport.read(1)

    def debug_sensors(self):
        """Sends command to get the current state of the sensors, used for debugging sensor issues
        :return: str
            String representing the current value of the sensors
        """
        if self.comport.isOpen():
            self.comport.write('CP0E'.encode('ascii'))
        return read_dispenser_line(self)

    def wheel_test(self):
        """Sends command to run the wheel test procedure
        Try breaking the eyelet sensor to verify operation or updating the motor speed to change the max speed
        :return: None
            Loop is terminated by the dispenser sending a '0' for successful completion
        """
        if self.comport.isOpen():
            self.comport.write('CF0E'.encode('ascii'))
            wheel_out = ""
            while wheel_out != b'0':
                wheel_out = read_dispenser_line(self)
                if wheel_out[-3:] != '127':
                    print(wheel_out)
                elif wheel_out[-3:] == '127':
                    print(wheel_out)
                    wheel_out = self.comport.read(1)

    def disconnect(self):
        """Disconnects the internal comport object.
        :return: none
        """
        self.comport.close()


def list_comports():
    """Wrapper utility to show exposed COM ports.
    :return: list
        List consisting of serial object representing the available COM ports
    """
    comports = serial.tools.list_ports.comports()
    print([comport.device for comport in comports])
    return comports


def read_dispenser_line(dispenser):
    """Wrapper function on serial.readline() function to handle Arduino Serial.println(). Use in loop to handle
    continuous output.
    Source: https://stackoverflow.com/questions/24074914/python-to-arduino-serial-read-write
    :param dispenser: CanineDispenser
        Pass a dispenser object with an open comport
    :return: str
        Trimmed Arduino output string
    """
    line_out = str(dispenser.comport.readline())
    return line_out[2:][:-5]


def dispenser_test(expected_dispensers):
    """This function is to be used to test multiple dispensers for basic functionality
    Parameters
    :param expected_dispensers: int
        The number of dispensers to be connected to
    :return: None
        Used to break while loop, returns None value
    """
```

```
    comports = list_comports()

    dispensers = []

    while expected_dispensers != 0:
        try:
            com_select = input("Select COM port from list above: ")
            if int(com_select) < 0 or int(com_select) > len(comports) - 1:
                print('Invalid input. Valid range is 0 to', len(comports) - 1)
            else:
                dispensers.append(CanineDispenser(comport_name=comports[int(com_select)].device))
                expected_dispensers -= 1
        except:
            print('Invalid COM port selected, use numerals in range [0, 9]. Returning control.')
            return

    while True:
        command = input('Enter number of treats to dispense [0, 9] or "exit" to quit: ')
        try:
            if command is 'exit':
                for dispenser in dispensers:
                    dispenser.disconnect()
                return
            elif int(command) < 0 or int(command) > 9:
                print('Invalid input. Valid range is [0, 9].')
            else:
                for dispenser in dispensers:
                    out = dispenser.dispense_treats(command)
                    if out == b'1':
                        print('Invalid command for', dispenser.comport.name)
                    elif out == b'0':
                        print('Successful command for', dispenser.comport.name)
        except:
            print('Invalid input. Only use numerals [0, 9] or type "exit". Returning control.')
            return


def dispenser_functionality_test(comport_name):
    """Sends all commands that return a byte return code.
    Parameters
    :param comport_name: str
        Name of the COM port, i.e. '/dev/ttyACM0' or 'COM31'
    :return: None
        Prints state of the commands to command line
    """
    dispenser = CanineDispenser(comport_name=comport_name)
    D_out = dispenser.dispense_treats(1)
    A_out = dispenser.dispense_treats_multi(1)
    M_out = dispenser.change_motor_speed(127)
    RF_out = dispenser.update_rf_amount(1)

    if D_out == b'0' and A_out == b'0' and M_out == b'0' and RF_out == b'0':
        print('D: ', D_out, '\nA: ', A_out, '\nM: ', M_out, '\nRF: ', RF_out, '\nDispenser operating
successfully.')
    else:
        print('D: ', D_out, '\nA: ', A_out, '\nM: ', M_out, '\nRF: ', RF_out, '\nDispenser not operating
correctly.')
```

Document Revision History

Revision, Name, and Date	Revision Description
V0.1, Walker Arce, 21 March 2020	First draft creation date for Treat & Train retrofitted circuit.
V1.0, Walker Arce, 27 April 2020	Added documentation on custom board
V1.1, Walker Arce, 28 April 2020	Added documentation on updated software
V1.2, Walker Arce, 12 May 2020	Added docs on new commands and Raspberry Pi
V1.3, Walker Arce, 16 May 2020	Added build instructions and bills of material
V1.4, Walker Arce, 17 May 2020	Added Python source, updated interfacing, and commands