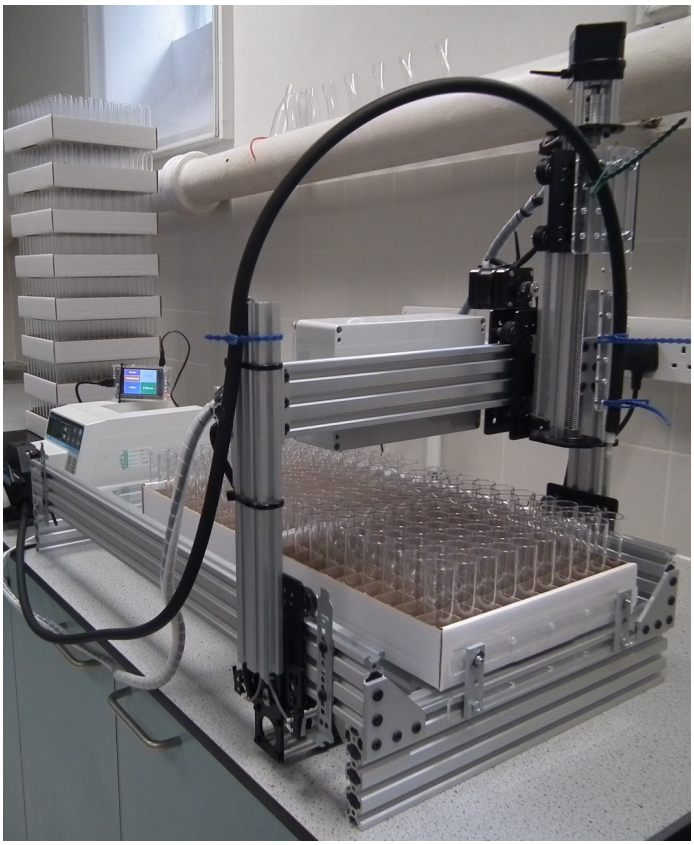
CNC Fly Food Dispenser

Matt Wayland

2024-01-19

Contents

Preface



Github

WaylandM/fly-food-robot

License

Contact

Matt Wayland

Publication

Article in Journal of Open Hardware: https://openhardware.metajnl.com/articles/10.5334/joh.9/

Colophon

This book was produced using the **bookdown** package (Xie, 2017), which was built on top of R Markdown and **knitr** (Xie, 2015).

Chapter 1

Introduction

The fruit fly, *Drosophila melanogaster*, is one of the most important model organisms in biological research. Maintaining stocks of fruit flies in the laboratory is labour-intensive. One task which lends itself to automation is the production of the vials of food in which the flies are reared. Fly facilities typically have to generate several thousand vials of fly food each week to sustain their fly stocks. The system presented here combines a Cartesian coordinate robot with a peristaltic pump (Figure 1.1). The design of the robot is based on the Routy CNC Router created by Mark Carew (http://openbuilds.org/builds/routy-cnc-router-v-slot-belt-pinion.101/), and uses belt and pully actuators for the X and Y axes, and a leadscrew actuator for the Z axis. CNC motion and operation of the peristaltic pump are controlled by grbl (https://github.com/gnea/grbl), an open source, embedded, high performance g-code parser. Grbl is written in optimized C and runs directly on an Arduino. A Raspberry Pi is used to generate and stream G-code instructions to Grbl. A touch screen on the Raspberry Pi provides a graphical user interface to the system. The system has capacity to fill two boxes of fly food vials at a time. Instructions for building the hardware are available on DocuBricks.

Software installation and configuration on the Arduino and raspberry pi are detailed in chapters 2 and 3. Chapter 5 explains how to program the robot to fill vials of fly food. Instructions for the routine use of the robot are provided in chapter 6. To see a video of the robot in action, go to:

https://doi.org/10.6084/m9.figshare.5175223.v1

An article providing an overview of this project has been published in the Journal of Open Hardware: https://openhardware.metajnl.com/articles/10.5334/joh.9/

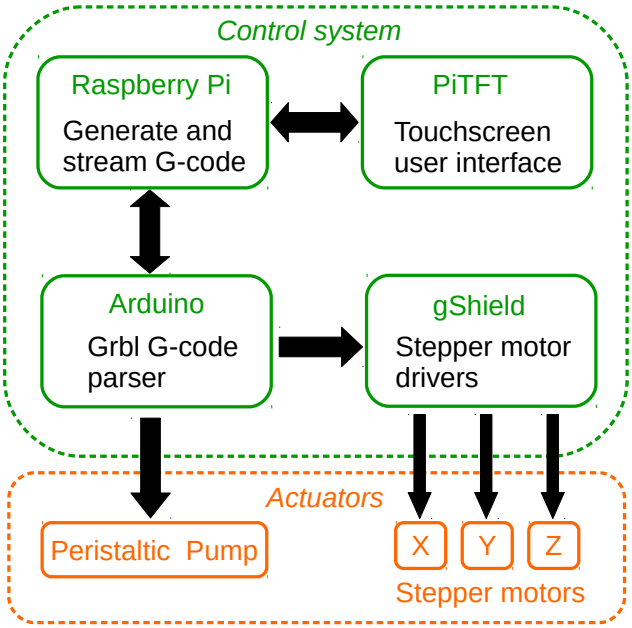


Figure 1.1: System architecture.

Chapter 2

Grbl installation and configuration

2.1 Overview

CNC motion control is provided by grbl (https://github.com/gnea/grbl), an open source, embedded, high performance g-code parser. Grbl is written in optimized C and runs directly on an Arduino. This is used in conjunction with the gShield (formerly known as grblshield) which provides the hardware drivers for the stepper motors. Grbl sends out TTL signals on pins A3 and 13 of the Arduino to control coolant flow and spindle direction, respectively. Here these signals are used to remotely control a peristaltic pump.

2.2 Flashing Grbl to Arduino

To flash Grbl to the Arduino you will need a computer with the latest version of the Arduino IDE installed. The following instructions for flashing Grbl to the Arduino are taken from: https://github.com/gnea/grbl/wiki/Compiling-Grbl

NOTE: Before starting, delete prior Grbl library installations from the Arduino IDE. Otherwise, you'll have compiling issues! On a Mac, Arduino libraries are located in `~/Documents/Arduino/libraries/`. On Windows, it's in `My Documents\Arduino\libraries`.

1. Download the Grbl source code.

- Open the following page in your web browser: https://github.com/gnea/grbl
- Click on the `<>Code` Tab

- Click the `Clone or Download` green button on the Grbl home page.
- Click the `Download ZIP`
- Unzip the download and you'll have a folder called `grbl-XXX`, where `XXX` is the release version.

2. Launch the Arduino IDE (figure 2.1)

- Make sure you are using the most recent version of the Arduino IDE!

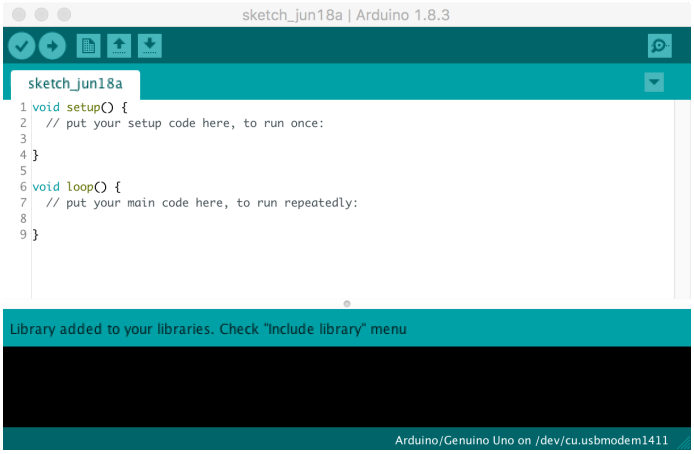


Figure 2.1: Arduino IDE

3. Load Grbl into the Arduino IDE as a Library.

- Click the `Sketch` drop-down menu, navigate to `Include Library` and select `Add .ZIP Library`.
- **IMPORTANT:** Select the `Grbl` folder ***inside*** the `grbl-XXX` folder, which **only** contains the source files and an example directory (figure 2.2).
- If you accidentally select the `.zip` file or the wrong folder, you will need to navigate to your Arduino library, delete the mistake, and re-do Step 3.

4. Open the `GrblUpload` Arduino example.

- Click the `File` drop-down menu, navigate to `Examples->Grbl`, and select `GrblUpload` (figure 2.3).

5. Compile and upload Grbl to your Arduino.

- Connect your computer directly to the Arduino using the USB cable. Unplug the USB cable from the raspberry pi and plug it into your computer. Remember to reconnect the raspberry pi to the Arduino after you have finished configuring Grbl (figure 2.4).
- Make sure your board is set to the Arduino Uno in the `Tool->Board` menu and the serial port is selected correctly in `Tool->Serial Port`.

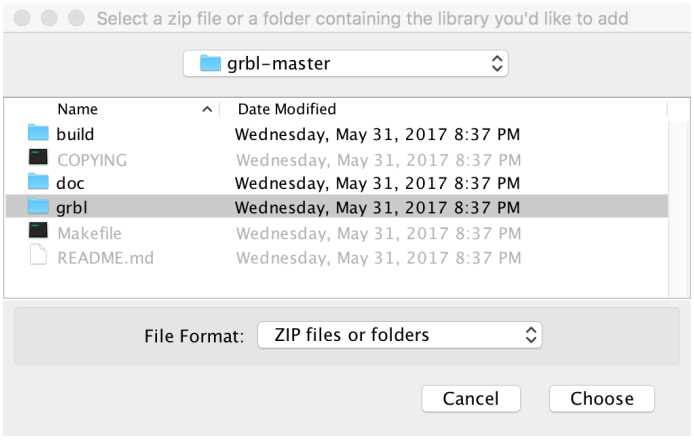


Figure 2.2: Loading Grbl library into the Arduino IDE

- Click the `Upload`, and Grbl should compile and flash to your Arduino! (Flashing with a programmer also works by using the `Upload Using Programmer` menu command.)

2.3 Check serial connection to Grbl

NOTE: Before powering up the gShield and motors, check that the actuator carriages for all three axes are approximately centred (figure 2.5). Initially we do not know in which direction the actuator carriages will travel when G-code commands are issued, so positioning each in the middle of its range reduces the risk of collisions with the end stops.

1. Open serial monitor in Arduino IDE

- Click `Tools` drop-down menu, and select `Serial Monitor`
- Note that line-ending is set to `Carriage return` and baud rate is set to `115200` (figure 2.6)

2. Try issuing a G-code command.

- Type `?` and hit return.
- This command will report the current position; as we have just started the system up all axes will be at 0.000.

3. Now try moving actuators

- To move in the x-axis type `x5` and hit return. Make a note of the direction in which the actuator carriage moves. N.B. this command tells Grbl to

Figure 2.3: GrblUpload example file

Figure 2.4: Laptop connected directly to Arduino

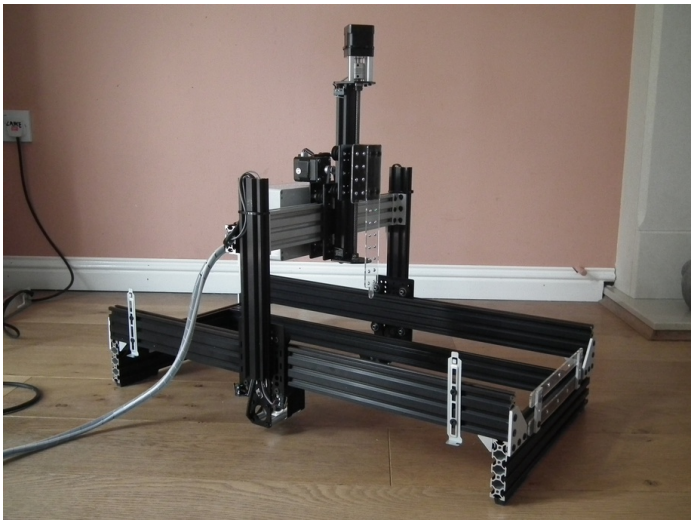


Figure 2.5: Actuator carriages centred in preparation for powering-up motors for first time

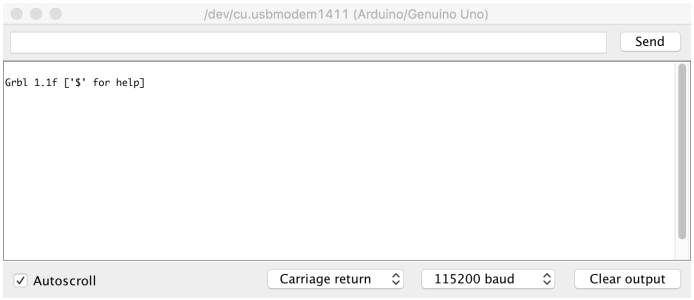


Figure 2.6: Arduino IDE Serial Monitor

move to the x coordinate that is 5 units from the origin, it is not equivalent to telling the robot to move 5 units in the x-axis.
- To move in the opposite direction along the x-axis type `x-5` and hit return.
- To return to the starting point, use `x0`
- Repeat for the other axes, replacing the x in the commands with y or z. Make a note of the direction the actuator carriages move with each command.

2.4 Grbl configuration

2.4.1 Read current configuration

- The `$$` command will report Grbl's current configuration.
- Descriptions of these settings can be found here: https://github.com/gnea/grbl/wiki/Grbl-v1.1-Configuration
- These settings will be modified in subsequent steps.

2.4.2 Check directionality of each axis.

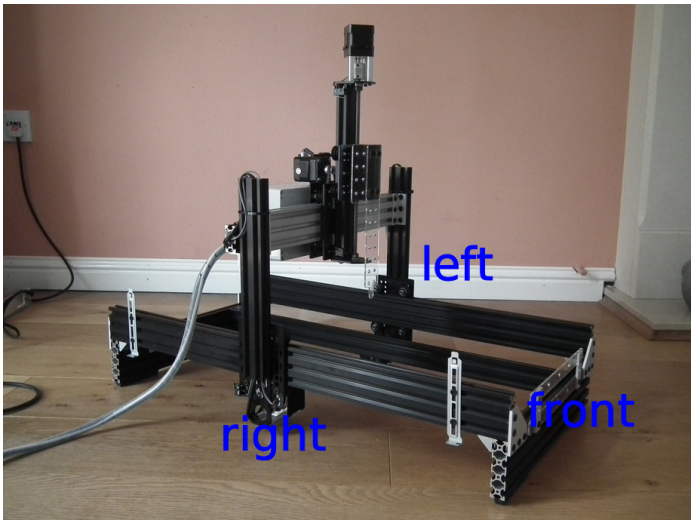


Figure 2.7: Orientation of robot.

- Orientation of the robot is shown in figure 2.7.
- At present the origins of all three axes are mid-way along each actuator, because this was the position of the actuator carriages when the system was started.

Table 2.1: Masks for direction port inversion.

Setting Value	Mask	Invert X	Invert Y	Invert Z
0	00000000	N	N	N
1	00000001	Y	N	N
2	00000010	N	Y	N
3	00000011	Y	Y	N
4	00000100	N	N	Y
5	00000101	Y	N	Y
6	00000110	N	Y	Y
7	00000111	Y	Y	Y

- Make sure actuator carriages are at their current origin by entering this command: `x0y0z0`
- Enter the command: `x5`. The x-axis carriage should move from right to left (orientation of robot is shown in figure 2.7. If it doesn't, make a note that it will need to be inverted.
- Enter the command: `y5`. The y-axis carriage should move forwards. If it doesn't, make a note that it will need to be inverted.
- Enter the command: `z5`. The z-axis carriage should move up. If it doesn't, make a note that it will need to be inverted.
- The direction of the actuators can be inverted using setting **$3**, the Direction port invert (mask). An appropriate value is selected from table 2.1. For example, to invert the direction of the X and Z axis actuators use the following command: `$3=5`

2.4.3 Activate hard limits

Hard limits are a safety feature to prevent the machine from travelling beyond the limits of travel. Grbl monitors the paired limit switches on each axis and if a switch is triggered it will immediately switch off all motors. Hard limits are activated by setting **$21** hard limits (boolean) to 1:

```
$21=1
```

2.4.4 Setup homing

The homing cycle is used to set the origin of the Cartesian coordinate system used by the robot. During the homing cycle Grbl moves each actuator in the positive direction until the limit switches are triggered. The homing cycle is activated by setting **$22** homing cycle (boolean) to 1:

`$22=1`

Initiate a homing cycle using the following command: `$h`. All actuator carriages should move to the origin of their axes. The origin of the Cartesian coordinate system (home) for the robot is shown in figures 2.8 and 2.9

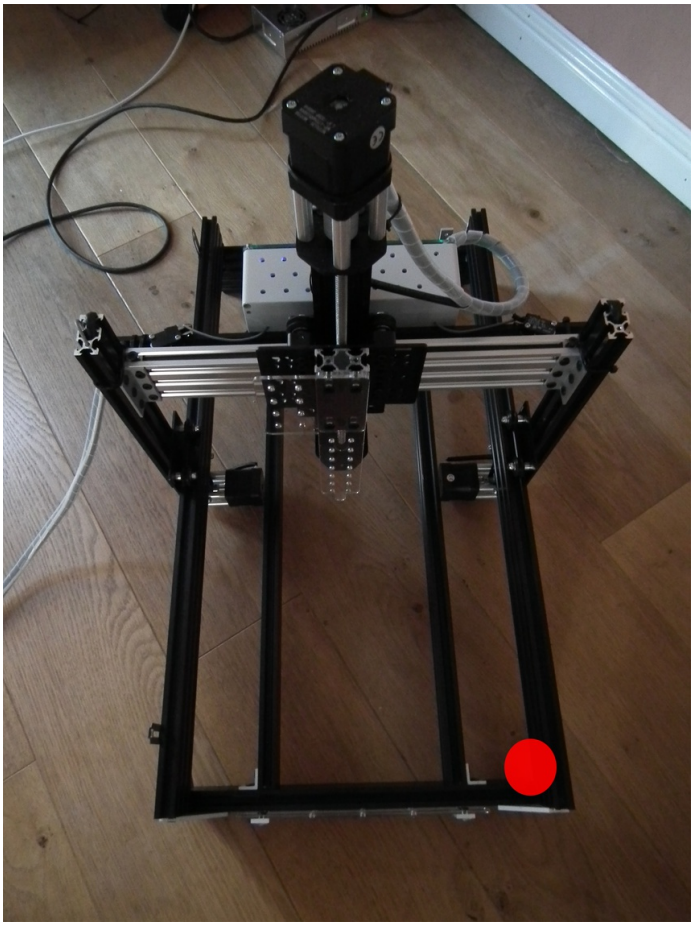


Figure 2.8: Origin of XY coordinate system.

We also need to set **$24** homing feed rate and **$25** homing seek rate. Homing seek rate is the initial speed at which Grbl searches for the limit switches. Once it has them, it makes slower approach at the homing feed rate to get a more precise location for machine zero. We will set homing seek rate to 1000 mm/min `$24=100` and homing feed rate to 100 mm/min `$25=1000`.

At the end of a homing cycle each actuator carriage must be moved off its home limit switch, otherwise the hard limit will be triggered. The **$27** Homing pull-off (mm) specifies the distance required to clear the limit switches. For our robot

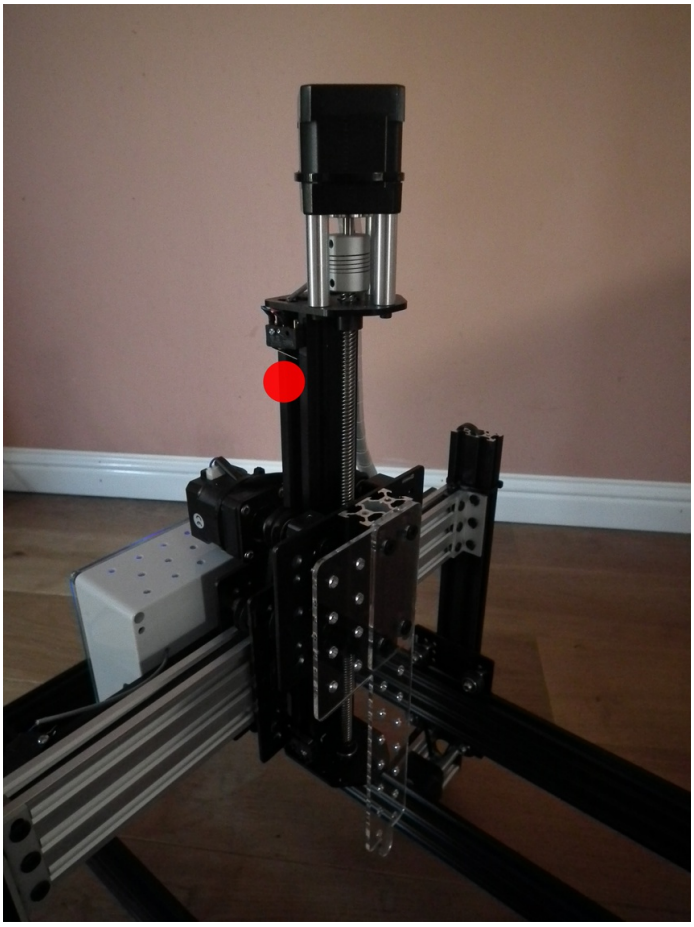


Figure 2.9: Origin of Z axis.

we will use a value of 5mm:

```
$27=5
```

2.4.5 Motor step size

$100, **$101** and **$102** define [X,Y,Z] steps/mm. Suitable values for our stepper motors are:

```
$100=40
$101=40
$102=49.673
```

If you are using a different type of stepper motor, the step size can be easily calculated by measuring how far each actuator moves in response to a G-code command. For example, we would calculate the step size for the X actuator as follows.

1. Using the serial monitor in the Arduino IDE, issue the following command to *home* the machine:

```
$h
```

2. Read the current position of the machine, as reported by the Grbl controller:

```
?
```

After homing the Cartesian coordinates should be zero minus the homing pull-off: * x = -5 * y = -5 * z = -5

3. Make a note of the physical position of the nozzle.

4. Issue a g-code command to move to the -100mm position on the x-axis:

```
x-100
```

Keep decreasing the value of x until the x-actuator carriage almost meets the limit switch on the right hand side of the machine. The greater the distance moved, the more precise our calculation of step-size will be.

5. Query Grbl's machine coordinates:

```
?
```

6. Measure the physical distance travelled along the x-axis in millimetres.

7. Find **$100**, the currently configured step size for the x-axis:

```
$$
```

8. Calculate the correct value for step-size:

```
current_step_size = steps/mm in current configuration
grbl_start = start position reported by Grbl controller (mm)
grbl_end = end position reported by Grbl controller (mm)
physical_distance = physical distance moved by actuator (mm)

steps/mm = -(curr_steps_per_mm * (end_pos_grbl-start_pos_grbl)) / physical_distance
```

2.4.6 Feed rates and acceleration

$110, **$111** and **$112** set the maximum rates (mm/min) for the X, Y and Z actuators, respectively. We will use the following values:

```
$110=5000
$111=5000
$112=2500
```

Acceleration (mm/sec^2) is set to 50 for all axes:

```
$120=50
$121=50
$122=50
```

2.4.7 Summary of settings

```
$0=10
$1=25
$2=0
$3=5
$4=0
$5=0
$6=0
$10=1
$11=0.010
$12=0.002
$13=0
$20=0
$21=1
$22=1
$23=0
$24=100.000
$25=1000.000
$26=250
$27=5.000
$30=1000
$31=0
```

```
$32=0
$100=40.000
$101=40.000
$102=49.673
$110=5000.000
$111=5000.000
$112=2500.000
$120=50.000
$121=50.000
$122=50.000
$130=200.000
$131=500.000
$132=200.000
```

2.5 Setting motor current

The gShield has trimpots for adjusting the motor current of each axis, as shown in figure 2.10.

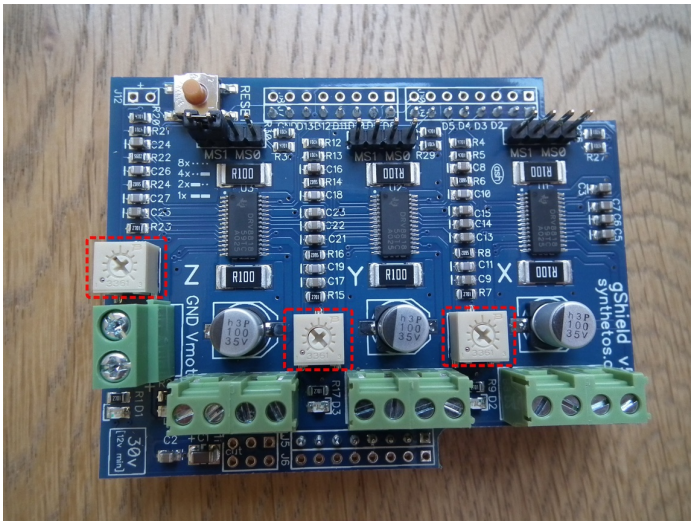


Figure 2.10: gShield trimpots

Instructions for setting motor current are provided here: https://github.com/synthetos/grblShield/wiki/Using-grblShield#setting-motor-current

Chapter 3

Raspberry Pi setup

3.1 Overview

The raspberry pi is used to generate, then stream g-code to the Arduino. A small thin-film-transistor (TFT) touchscreen serves as the user interface to the raspberry pi.

3.2 Install image

~~The first step is to install Adafruit's custom raspberry pi image on the micro SD card. The custom image is described here:~~

~~`https://learn.adafruit.com/adafruit-pitft-28-inch-resistive-touchscreen-display-raspberry-pi/easy`~~

~~We want the classic version which boots into X by default, rather then the lite version that boots to the command line. The classic version can be downloaded from this link:~~

~~`https://s3.amazonaws.com/adafruit-raspberry-pi/2016-10-18-pitft-28r.zip`~~

Adafruit's custom raspberry pi image is no longer available. I have created an image (pitft.img) based on Adafruit's original that is preconfigured for use with the robot. It can be downloaded from:

https://zenodo.org/records/10534021

Instructions on installing images on SD cards can be found here:

https://www.raspberrypi.org/documentation/installation/installing-images/

3.3 Network configuration

The small screen of the pitft makes using most applications quite tricky. Therefore the first thing we should do after installing the image is configure networking, so that we can access the raspberry pi remotely using ssh.

To set a static IP address for the ethernet adapter, edit the following lines of `/etc/dhcpcd.conf`:

```
interface eth0

static ip_address=192.168.1.3/24
static routers=192.168.1.254
static domain_name_servers=192.168.1.254
```

ip_address, **routers** and **domain_name_servers** should be set to values appropriate for your network.

To raspberry pi can then be accessed using ssh, *e.g.*:

```
ssh pi@192.168.1.3
```

The default password for the **pi** user account is **raspberry**

3.4 Install minicom

N.B. Minicom is pre-installed on my pitft.img

Minicom is useful for manual control of the robot and for editing grbl settings.

To install minicom run these two commands:

```
sudo apt-get update
sudo apt-get install minicom
```

Before we can use minicom we need to enable serial:

```
sudo nano /boot/config.txt
```

Change the last line of this file from

```
enable_uart=0
```

to

```
enable_uart=1
```

If you have not already done so, reconnect the raspberry pi to the arduino using the USB cable. To use minicom to connect to the grbl controller running on the arduino, use:

```
sudo minicom -D /dev/ttyACM0 -b115200
```

3.5 Expand filesystem

Expand filesystem on micro SD card:

```
sudo raspi-config
(expand filesystem)
sudo reboot
```

3.6 Install python scripts

N.B. The python scripts are pre-installed on my pitft.img

Make sure you are in pi's home directory:

```
cd
```

Download and unpack robot.tar.gz

```
curl -O https://raw.githubusercontent.com/WaylandM/fly-food-robot/master/raspberrypi/robot.tar.gz
tar xzvf robot.tar.gz
```

The **robot** directory contains two subdirectories: **nc** (g-code scripts) and **py** (python scripts).

To automatically launch the robot GUI when the raspberry pi starts up, we need to edit the autostart file for the pi user:

```
sudo nano /home/pi/.config/lxsession/LXDE-pi/autostart
```

Add the following line to autostart:

```
@/home/pi/robot/py/fly_gui.py
```

Chapter 4

System start-up

4.1 Attach Norprene tubing

1. Attach the nozzle end of the Norprene tubing to the holder on the Z-axis actuator using releasable cable ties (figure 4.1).
2. Attach the Masterflex Norprene tubing to the right-hand side vertical post using a releasable cable tie. Ensure there is a large loop in the tubing between this attachment point and the nozzle so that the x-axis actuator can move freely (figure 4.2)
3. Feed the tubing through the peristaltic pump (figure 4.3) and into your vat of fly food (figure 4.4).

4.2 Switch on all devices

- power supply unit for gShield and motors
- raspberry pi
- peristaltic pump

4.3 Prime pump

- Position a beaker under the nozzle (figure 4.5).
- Press and hold the prime button on the front of the peristaltic pump until a continuous stream of fly food is pumped into the beaker (figure 4.6)

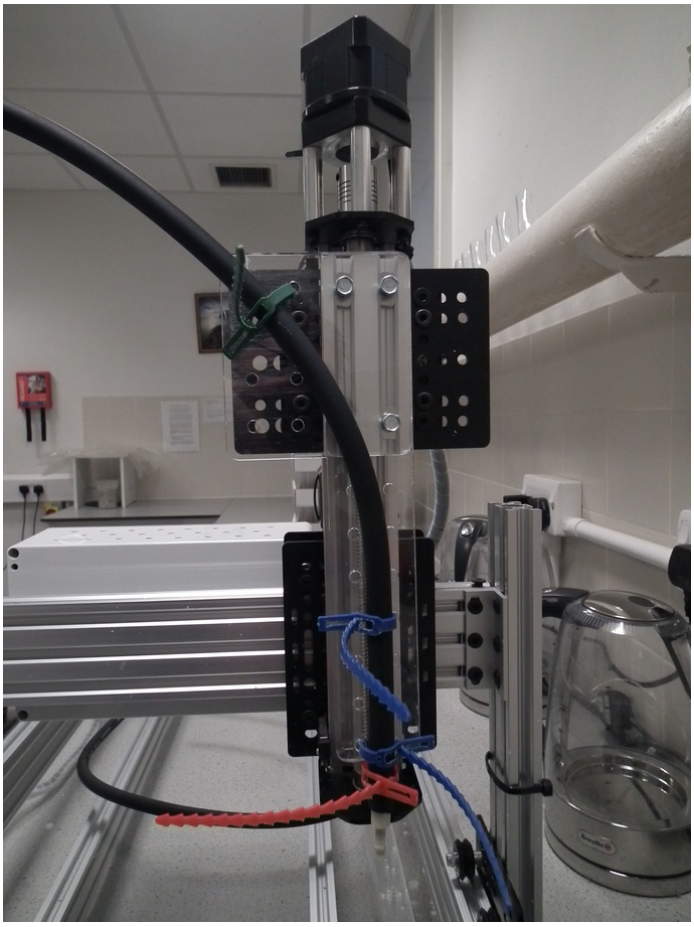


Figure 4.1: Attachment of nozzle end of tubing to the Z-axis actuator.

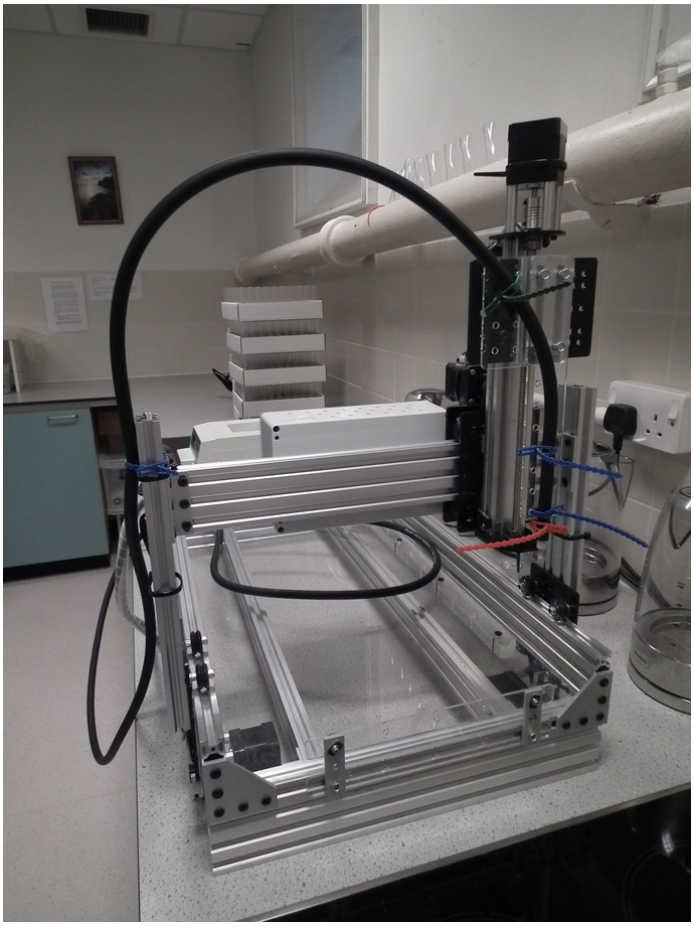


Figure 4.2: Attachment of Norprene tubing to the right-hand side vertical post.

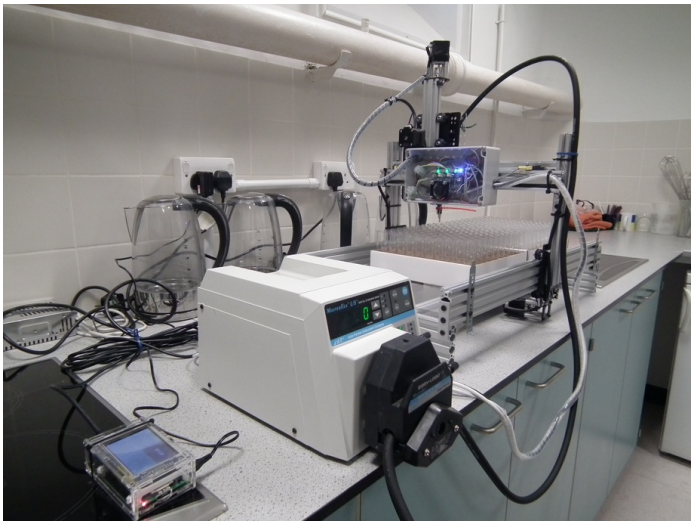


Figure 4.3: Norprene tubing passing through peristaltic pump.

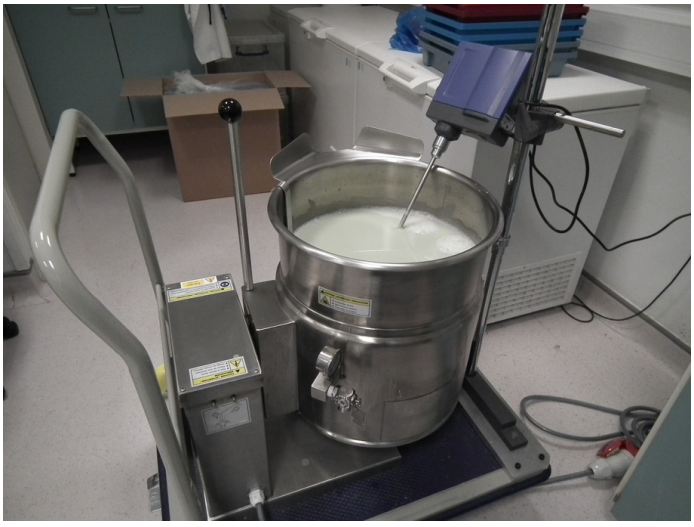


Figure 4.4: Vat of fly food.

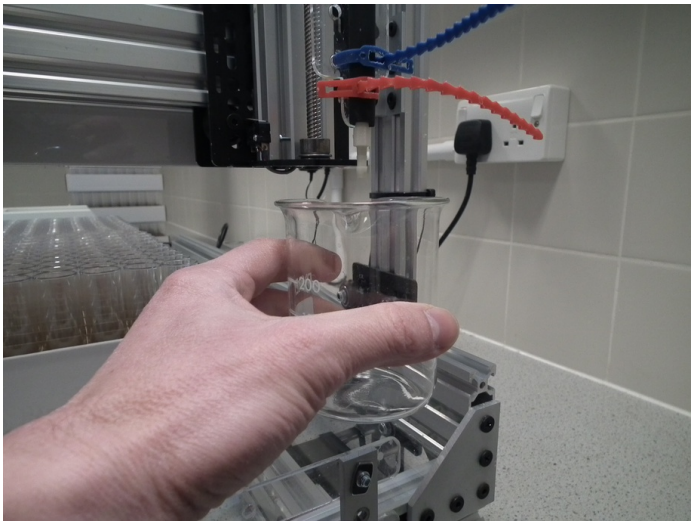


Figure 4.5: Positioning of beaker under nozzle to collect fly food expelled during priming of peristaltic pump.

Figure 4.6: Prime button on peristaltic pump.

Chapter 5

Program robot to fill vials

5.1 Overview

The movement of the robot is programmed in G-code. We only need nine G-code commands to control the robot (table 5.1).

A G-code program for filling vials of food could be created manually, by listing the necessary commands sequentially in a text file. However, this would be laborious and error prone. If the size of the boxes of vials are known, the G-code can be programmatically generated.

5.2 Load boxes

Load boxes onto the platform of the robot (figure 5.1). * The first box should be flush with the fence and the guide rail. * The second box should be flush with the first and the guide rail. * The boxes we are using have a pair of double-thickness side-walls and a pair of single-thickness side-walls. The pairs are on opposite sides of the box. With this type of box it is important to note the orientation of the boxes when the Cartesian coordinates of the vials are determined, because the same orientation must be used when filling vials with food. We load the boxes with a double-thickness side-wall facing forwards.

5.3 Determine box coordinates

We need to determine the Cartesian coordinates of vials in diagonally opposite corners of each box.

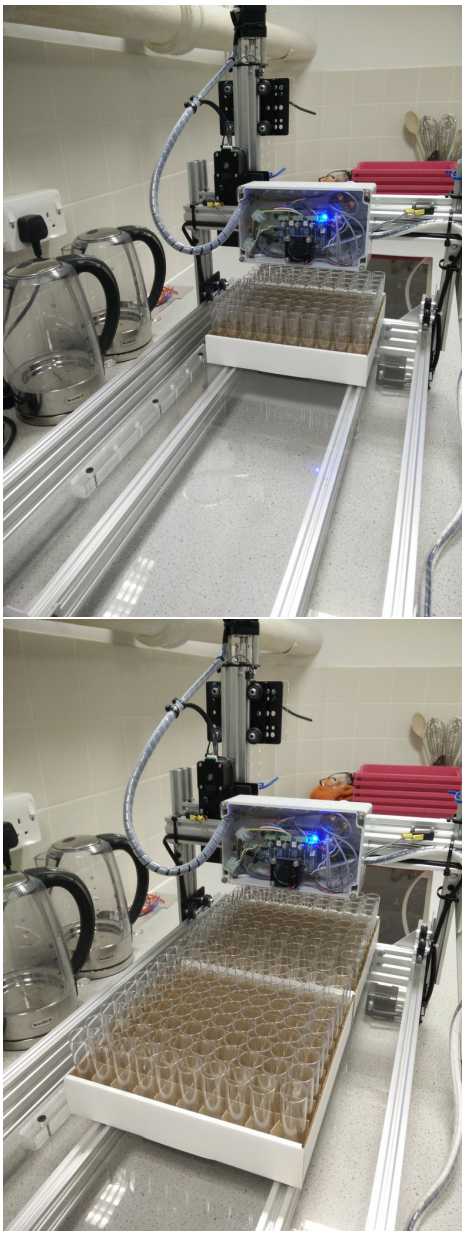


Figure 5.1: Loading boxes of vials.

Table 5.1: G-code commands used to control robot.

Code	Description
x	absolute position of x-axis
y	absolute position of y-axis
z	absolute position of z-axis
g4	dwell time (control parameter p specifies seconds)
m3	set pump rotation to clockwise
m4	set pump rotation to counter clockwise
m8	start pump
m9	stop pump
$h	initiate homing cycle

1. Login to raspberry pi using ssh. My raspberry pi has the IP address 192.168.1.3 and so I would use:

- `ssh pi@192.168.1.3`
- Default password for the **pi** user account is 'raspberry'.

2. Use minicom to connect to the Grbl controller running on the Arduino, so that we can interactively control the robot from the command line:

```
sudo minicom -D /dev/ttyACM0 -b115200
```

3. Make sure nozzle is at ** home ** position by issuing homing command:

```
$h
```

4. First we will determine the Cartesian coordinates of the vial in the front left corner of the first box.

- Make small movements in X and Y until the nozzle is centred over this vial, *e.g.*:

```
x-8 y-8
```

- Lower the nozzle in small increments until it is just 2-3mm above the the top of the vial (figure 5.2), *e.g.*:

```
z-20
```

- Query the current X, Y and Z coordinates by issuing the following command:

```
?
```

Make a note of all three coordinates.

5. Issue G-code commands to move the nozzle laterally until it is over the back right vial of the first box (figure 5.3).

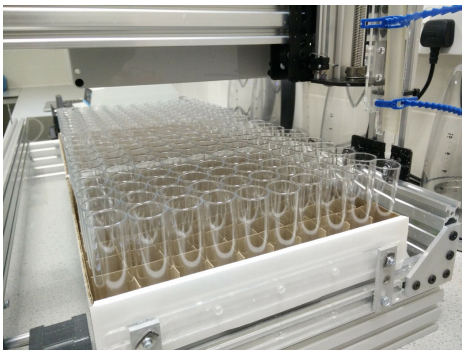


Figure 5.2: Nozzle positioned over the front left vial in box 1.

Figure 5.3: Nozzle positioned over the back right vial in box 1.

- Use `?` command to query nozzle position, and make a note of the X and Y coordinates:

6. Move the nozzle laterally until it is over the front left vial of the second box (figure 5.4), then record X and Y coordinates.

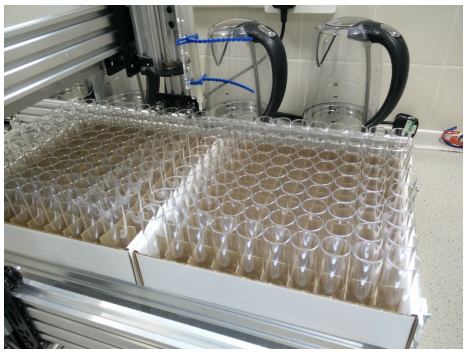


Figure 5.4: Nozzle positioned over the front left vial in box 2.

7. Finally determine the X and Y coordinates of the back right vial in the second box (figure 5.5).

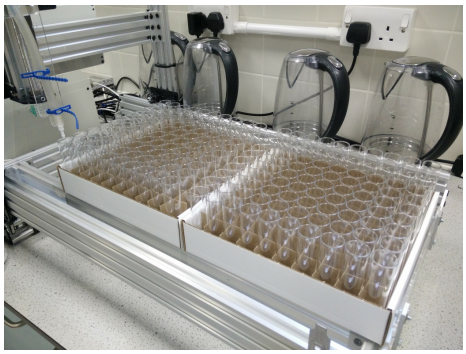


Figure 5.5: Nozzle positioned over the back right vial in box 2.

5.4 Calibrate pump

The peristaltic pump is started and stopped using the `m8` and `m9` G-code commands, respectively (table 5.1). To maximize speed, the pump will be run at its maximum flow rate of 30ml/second. In our fly facility, we add 9ml of food to each vial, therefore based on the maximum flow rate, we should only need to run the pump for 0.3 seconds to dispense 9ml of food. However, there is

latency in the system and the pump does not reach its maximum flow rate instantaneously on activation. Therefore, it is important to determine the *fill time* empirically. We do this by programming the robot to test fill a single box of vials using a range of *fill times*. The **calibrate_pump.py** script downloaded to the Raspberry Pi in stage 3.6 can be used to generate the appropriate G-code program.

1. Open **calibrate_pump.py** for editing:

```
sudo nano /home/pi/robot/py/calibrate_pump.py
```

2. Near the top of the file (line 32 onwards) are various settings to be modified:

```
# SETTINGS

# filename
filename = '/home/pi/robot/nc/calibrate_pump.nc'

# home/datum + homing pull-off (mm) (value of Grbl setting $27)
x_home = -5
y_home = -5
z_home = -5

# z value providing minimal clearance between nozzle and top of vials
z_fill = -62

# peristaltic pump settings
min_fill_time = 0.3
max_fill_time = 0.6
# pause to allow for drips before moving to next vial
drip_pause = 0.1

# vial coordinates (x,y)
frontLeft = (-8,-14)
backRight = (-236,-240)

nrows=10
ncols=10
```

- The **filename** is the full path to the G-code file that will be generated by the python script.
- The parameters **x_home, y_home** and **z_home** are the Cartesian coordinates of the home position (*i.e.* home/datum + homing pull-off (mm)).
- Modify **z_fill** to the appropriate nozzle height for filling vials (this was determined in step 5 of section 5.3).
- The **min_fill_time** should be set to our estimate of *fill time* based on

Table 5.2: Example calibration fill times.

Box Row	Fill Time
1	0.30
2	0.33
3	0.37
4	0.40
5	0.43
6	0.47
7	0.50
8	0.53
9	0.57
10	0.60

the pump's specified flow rate. We will set **max_fill_time** to twice the value of **min_fill_time**.

- The vial coordinates (**frontLeft** and **frontRight**) are those determined for the front left and back right vials in the first box (refer to steps 5 and 6 of section 5.3).
- Our boxes have ten rows, each containing ten vials, so we set **nrows** and **ncols** to 10.

3. After editing the settings in **calibrate_pump.py**, run the script to generate a G-code program:

```
./robot/py/calibrate_pump.py
```

This will generate a G-code program: **/home/pi/robot/nc/calibrate_pump.nc** which will iteratively increase the fill time for each successive row of vials. The **min_fill_time** will be used for the first row of vials and the fill time will be increased by (**max_fill_time** - **min_fill_time**) / (**nrows** - 1.0) for each successive row (*e.g.* table 5.2). Note that the fill times are rounded to the nearest 100th of a second. The fill time for each row is listed in the comments at the top of the **/home/pi/robot/nc/calibrate_pump.nc** file; to view this information run:

```
head -n12 ~/robot/nc/calibrate_pump.nc
```

4. Send the G-code pump-calibration program to the Grbl controller **/home/pi/robot/py/stream2.py** can be used to stream a text file of G-code commands to Grbl:

```
./robot/py/stream2.py robot/nc/calibrate_pump.nc /dev/ttyACM0
```

Once the robot has completed the calibration run and returned the nozzle to the home position, inspect the fill level in each row of vials. Identify the row in which vials are filled with the desired volume of food and then refer to the

/home/pi/robot/calibrate_pump.nc file to find out the fill time used for that particular row.

5.5 Generate programs

Once the pump has been calibrated we are ready to generate the G-code instructions for the routine filling of vials. The **fill_boxes.py** script downloaded to the Raspberry Pi in stage 3.6 is used to generate two G-code programs:

- **1_box.nc** - fill one box of vials
- **2_boxes.nc** - fill two boxes of vials

1. Open **fill_boxes.py** for editing:

```
sudo nano /home/pi/robot/py/fill_boxes.py
```

2. Near the top of the file (line 32 onwards) are various settings to be modified:

```
# SETTINGS
# modify values of variables in this section to match your system

# filenames
filename1Box = '/home/pi/robot/nc/1_box.nc'
filename2Boxes = '/home/pi/robot/nc/2_boxes.nc'

# home/datum + homing pull-off (mm) (value of Grbl setting $27)
x_home = -5
y_home = -5
z_home = -5

# z value providing minimal clearance between nozzle and top of vials
z_fill = -62

# peristaltic pump settings
fill_time = 0.43
# pause to allow for drips before moving to next vial
drip_pause = 0.1

# vial coordinates (x,y)
box1FrontLeft = (-8,-14)
box1BackRight = (-236,-240)
box2FrontLeft = (-8,-286)
box2BackRight = (-236,-513)
```

```
nrows=10
ncols=10
nVials=nrows*ncols
```

- **filename1Box** and **filename2Boxes** specify the filename and full path to the two G-code programs that will be generated. You should not need to modify these values. The GUI script (**/home/pi/robot/py/fly_gui.py**) expects the two programs to have the default names and paths; if you alter the names or paths of these files, you will need to edit the GUI script.
- The parameters **x_home, y_home** and **z_home** are the Cartesian coordinates of the home position (*i.e.* home/datum + homing pull-off (mm)).
- Modify **z_fill** to the appropriate nozzle height for filling vials (this was determined in step 5 of section 5.3).
- **fill_time** was determined in the previous section 5.4.
- The default value of **drip_pause** should be appropriate for most systems.
- The vial coordinates (**box1FrontLeft**, **box1BackRight**, **box2FrontLeft** and **box2BackRight**) were determined in section 5.3.
- Our boxes have ten rows, each containing ten vials, so we set **nrows** and **ncols** to 10.

3. After editing the settings in **fill_boxes.py**, run the script to generate the two G-code programs:

```
./robot/py/fill_boxes.py
```

Chapter 6

Routine operation

6.1 Prepare system

Instructions for starting the system are provided in chapter 4.

6.2 Fill boxes

Load boxes onto platform as described in section 5.2.

From the touchscreen menu select **1 Box** or **2 Boxes** depending on the number of boxes on the platform (figure 6.1). The robot will **home** to zero the cartesian coordinate system, and then commence filling the vials.

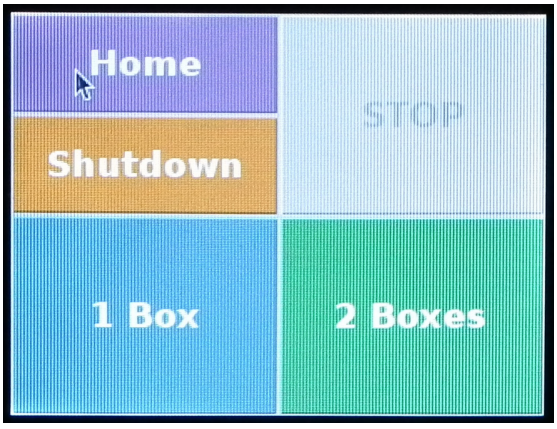


Figure 6.1: Touchscreen interface.

During filling the robot can be stopped by clicking on the **Stop** button (figure 6.2).

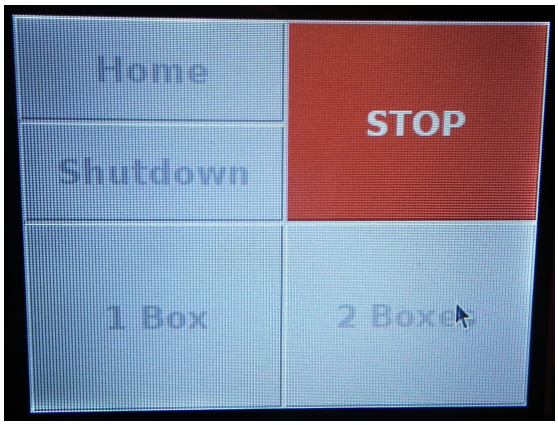


Figure 6.2: Appearance of touchscreen interface when a job is running.

6.3 Shutdown

- The raspberry pi should be shutdown using the **shutdown** button on the touchscreen.
- The Norprene tubing should be removed from the vat of fly food, then any food remaining in the tubing can be expelled by pressing and holding the **prime** button on the peristaltic pump. Ensure there is a receptacle under the nozzle to catch the expelled fly food. The Norprene tubing should be removed from the robot for cleaning/sterilization.
- All devices (raspberry pi, power supply unit for gShield and motors, and peristaltic pump) should be switched off.
- The robot can be wiped clean if necessary using a damp cloth. The platform can be removed for thorough cleaning.

Bibliography

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2017). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.4.