Single Entry Module

Figure 1, picture of SEM (and/or CAD drawing?)

General description

These build instructions are for a generic 'single entry module' (SEM) which we have incorporated in larger builds not explained here. Therefore, these instructions are reduced and if the reader follows them only they will end up with a device that has an empty slot for a mouse cage connected by the SEM to an empty, non-walled area where the mouse would be passed. Therefore, for actual experiments, the module needs to be installed together with a behavioural measurement device, such as an open field as the simplest case.

The SEM (Figure 1) consists of a 200 mm long U-shaped corridor, constructed from a sheet of transparent, firm, light-weight plastic (made from two overhead projector transparency sheets) which is secured into a U-shape via a 3D-printed skeleton (PLA). The U-shaped corridor is screwed onto a 100 g load cell (sparkfun TAL221 SEN-14727) which is secured to a large acrylic floor plate. Laser cut 3 mm acrylic sheet pieces are used to construct edge guides for the two doors by chemical welding using chloroform (using appropriate PPE). Additional pieces of acrylic sheet are attached to the door edge guides to support the edges of the U-shaped corridor in case an animal pushes against it. The SEM can be closed off on both ends via vertical sliding doors constructed from 300 mm X 50 mm rectangles of 0.5 mm thick aluminium sheet which can be lowered into the floor of the maze via a servo motor attached to a beam below the maze. An RFID-tag reader (sparkfun XXX) is placed below the U-shaped

corridor, near door 2. Two beam break devices (BB), constructed from a xx nm laser and photoresistor circuit mounted on 3D printed ball joints are used to ensure safe operation of the doors. One (BB1) is directed 10 mm above door 1 when it is open. It is used to prevent the door 1 closing with an animal on top. This beam is unbroken when the first door is open and no animal is sitting in the opening. Another (BB2) is directed through the corridor at a safe distance (>100 mm) from door 2. A break in this beam signals exit through the module back to the home cage.

Operation of the module is as follows and corresponds to the MODEs in Raspberry Pi code SEM_only.py:

1) SEM is open for entry from home cage
2) An animal is detected with the RFID reader. If the weight is in range of one animal and no animal is detected on top of door 1, door 1 is closed, trapping one animal in the SEM. Otherwise wait for next RFID detection.
3) The weight is determined by averaging 50 consecutive reads. If the weight is in range of one animal, door 2 is opened and a minimum time out (minimum_entry_time) is waited until exit detection ensues.
4) When the animal is returning to the home cage through the SEM, BB2 is broken, and consequently door 2 closes, door 1 opens and the SEM is open for the next entry (MODE 1).

Materials

Table 1, Bill of materials for the SEM.

Part	Count, vol. or length	Manufacturer	Manufacturer serial # or *.stl file	Approximate cost, EUR	Part REORDER LATER
Arduino nano	1				
Raspberry Pi 4b	1				
Servo motor	2				
Switch					
5V power supply	2				
wire	500 mm				
Breadboard	1	Bud industries	BB-32650-B	3.30	
Transparent film A4 pieces	2	Staedtler	10DT6F		
Chloroform*	1 ml				
300 mm X 50 mm rectangles of 0.5 mm thick aluminium sheet	2				door
3-d printed parts	1		scale1.stl	7.6 all parts	A
	1		scale2.stl		B
	1		scale3.stl		C
	2		scale4.stl		D

	4		bb_base.stl		E
	4		bb_base_nut.stl		F
	1		bb1_send.stl		G
	1		bb1_receive.stl		H
	1		bb2_send.stl		I
	1		bb1_receive.stl		J
	2		door_base.stl		K
	2		door_base_nut.stl		L
	2		servo_clamp.stl		M
	2		servo_bolt.stl		N
	2		servo_bolt_cap.stl		O
	1		rfid_base.stl		
3 mm Acrylic sheet parts	4		door_guide_back.svg		P
	4		door_guide_spacer.svg		Q
	4		sem_support.svg		R
	1		floor.svg		S
	2		door_lever.svg		T
1 mm Acrylic sheet parts	4		door_guide_front.svg		U
	2		prevent_burrowing.svg		V
20 mm square profile aluminum rail, 500 mm length	4				
20 mm square profile aluminum rail, 300 mm length	6				
20 mm square profile aluminum rail, 1000 mm length	2				
M6 cap screws, 30 mm length	6	McMaster-Carr	90128A266	9.2	
M6 grub screws, 30 mm length	8				
M6 Drop-In T-Nut	8	thorlabs	XE25T1/M		
M3 cap screws, 30 mm length	4	McMaster-Carr	91290A171	7.2	
M3 cap screws, 10 mm length	2	McMaster-Carr	91274A105	6.4	
M3 nuts	2				

Approx. total cost:

* Welding acrylic and 3D-printed PLA parts together with chloroform must be done safely in a well ventilated space, with minimal amounts of chloroform and using appropriate PPE including gloves and an appropriate mask.

The 3D-printed and laser cut part drawings can be found at https://github.com/MaheshKarnani/Switch_maze/tree/main/Modules_SM/SingleEntryModule

Useful tools: allen key set, a small screw driver, wire cutters, a soldering iron, tape, 1 ml syringe for the chloroform, personal protective equipment and epoxy glue.

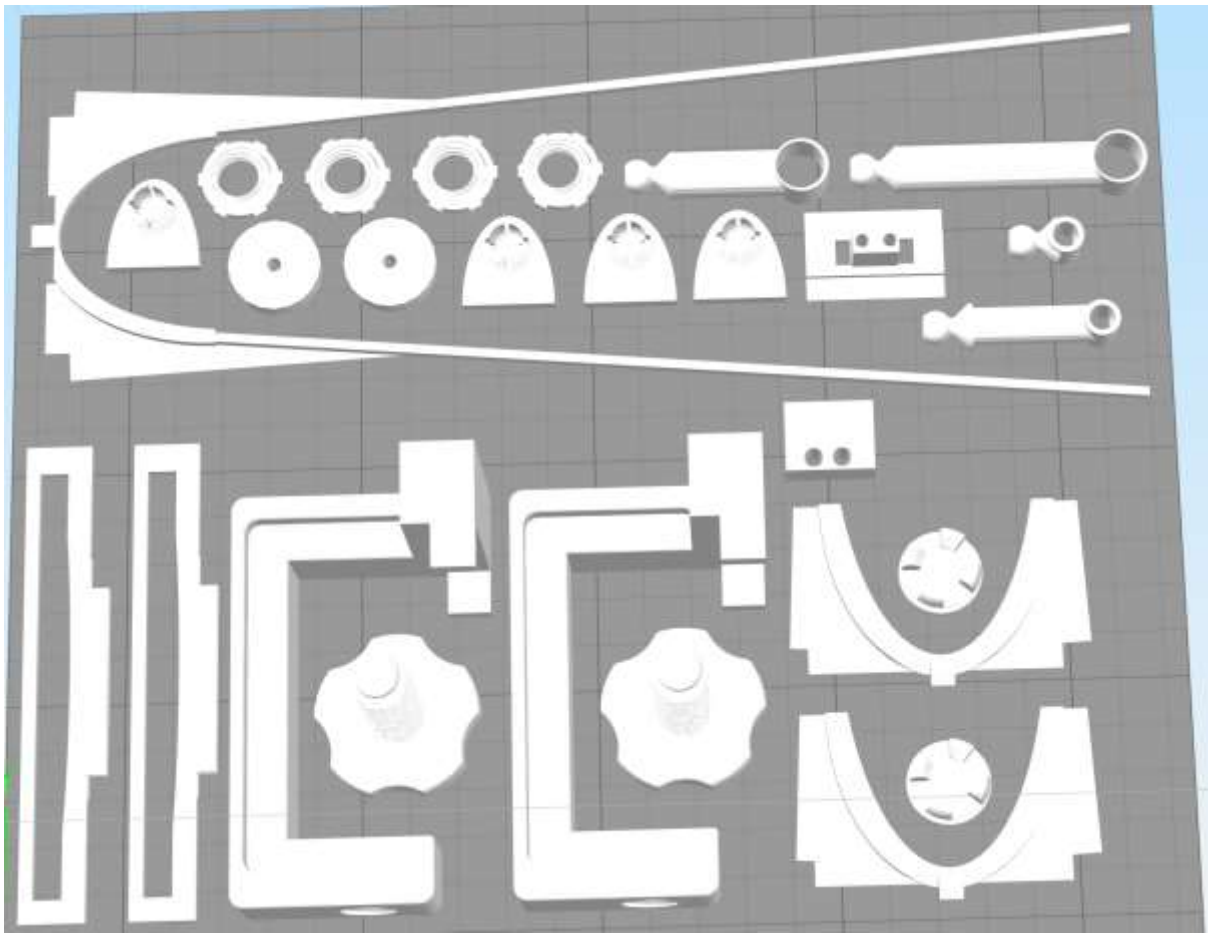


Figure 2, 3D printed parts.

Mechanical assembly

More assembly detail needed!

1) Build a support frame for the floor plate using 2x 300 mm, 2x 1000 mm and 4x 500 mm long 20 mm square profile aluminum rail according to Figure 3 and lay the 3mm acrylic sheet piece floor.svg on top (part S). A minimum height of 500 mm from the floor is recommended as the doors need to operate some distance below the floor.

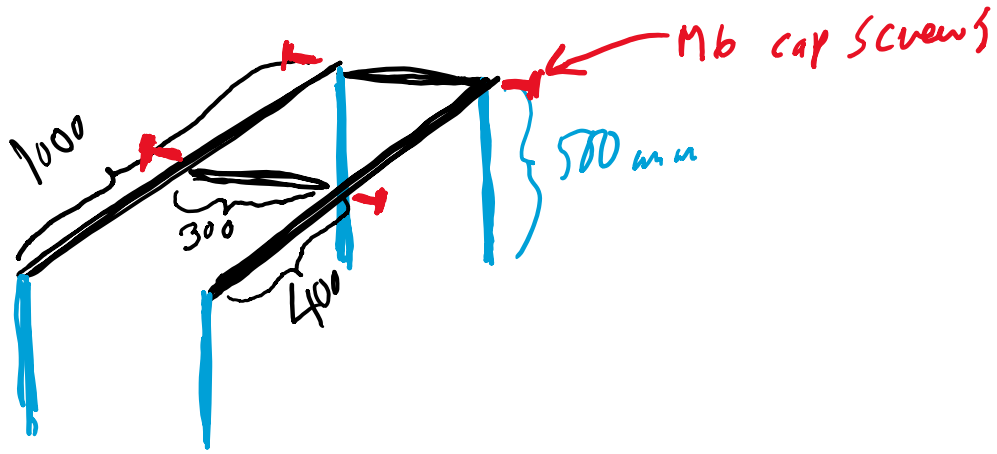


Figure 3. CAD of frame

2) Assemble the doors:

Using the drop-in T-nuts and M6 grub screws hang a 300 mm aluminum rail downward from the support frame, and attach another 300 mm aluminum rail perpendicular to that (Figure 4). Attach a servo motor on top of the perpendicular rail using 3D printed clamp (parts M-O).

Figure 4, servo motor installation.

Prepare the moving parts (Figure 5): Attach a lever made of 3 mm acrylic sheet (part T) to a servo hub (provided with the servo) using a 10 mm M3 screw. Once tightened, working in a well ventilated space/chemical hood add 1-2 drops of chloroform on the joint to activate the surfaces. After a few minutes add 2-3 drops of superglue to the joint. After the glue has set, unscrew the M3 screw. Slot a 300 mm X 50 mm rectangle of 0.5 mm thick aluminum sheet (part door) into a 3D printed base (part K) and add 2-3 drops of superglue to the joint. A small amount of epoxy glue can be used to bond these glued joints as they will carry variable loads. Attach a 30 mm M3 cap screw and nut to the other end of the lever.

Figure 5, acrylic door lever along with attachment pieces (above) and aluminum door (below).

Attach the lever to the servo such that its movement arc cannot collide with the floor. Slot the door through its aperture in the floor and attach it to the lever as shown in Figure 6 using the 3D printed nut (part L) to cap the 30 mm screw but do not tighten it against the door base.

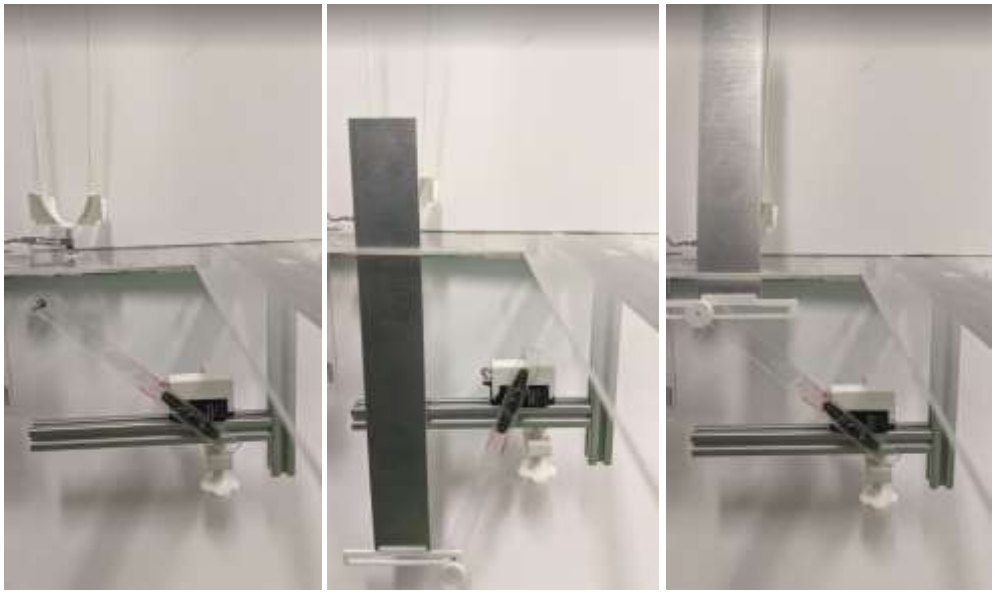


Figure 6, Door installation.

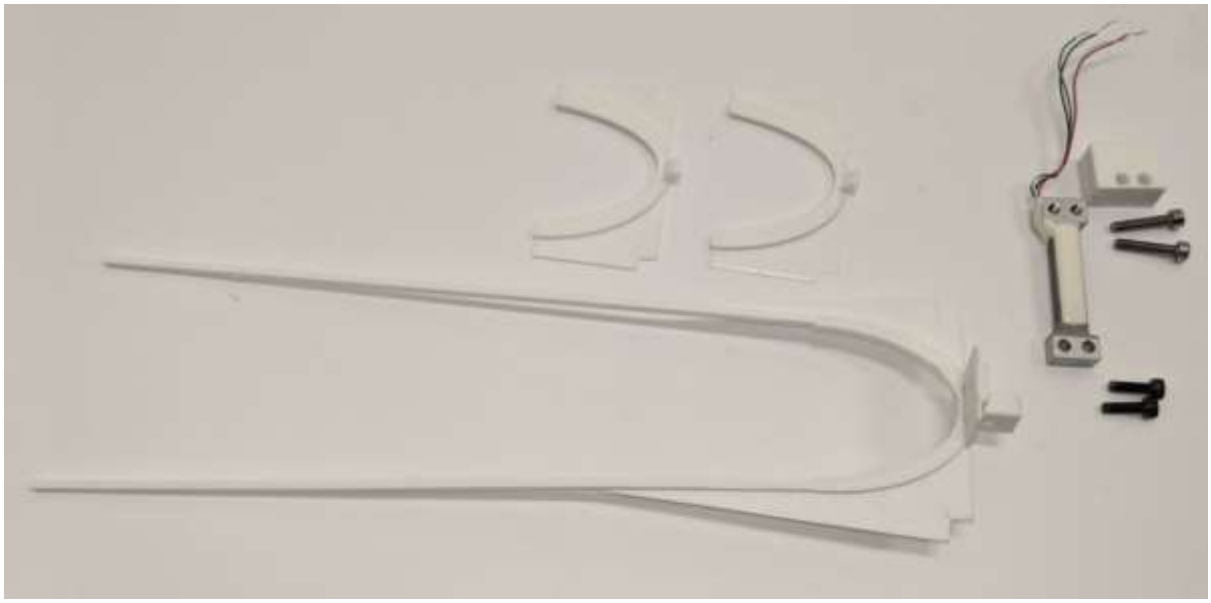
Guide frames for the door are constructed using acrylic sheet pieces P, Q and U according to figure 7. Clamp the pieces together with a binder clip and use a drop of chloroform to bond, making sure the

three pieces are aligned at the bottom for a strong bond to the floor plate. Make sure to leave enough room for the door to operate while remaining within the guide frame. Build frames for both doors.

Figure 7, guide frame installation. Use PPE and minimal amounts of chloroform.

3) Install the scale:

After guide frames for both doors are built, the U-shaped scale is assembled as follows. A scale backbone is assembled from 3D-printed parts A-D and a 100 g load cell (Sparkfun XXX) using M3 screws and push-fitting the pieces as shown in Figure 8. The prints may need to be filed down at the attachment points to get a good fit. A loose push-fit is acceptable at this stage as epoxy glue will be used later. The scale backbone is screwed in place firmly through the 10 x 3 mm slits in the floor plate.



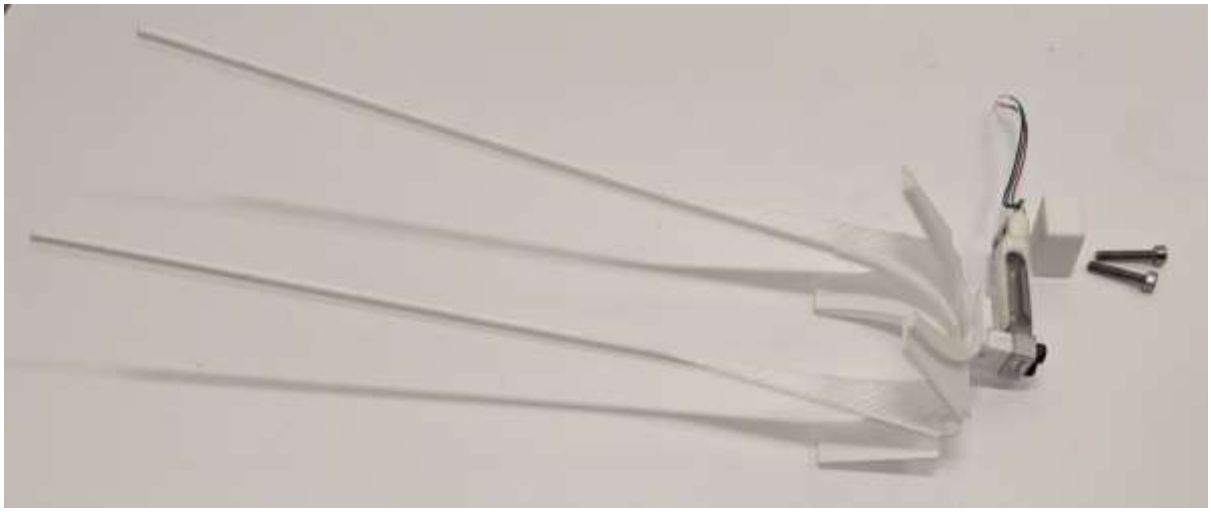


Figure 8, installing the scale backbone.

A 597 x 200 mm rectangle of transparency film is made by taking two A4 sheets, cutting one in half and taping the pieces to the ends of the intact one, followed by cutting 10 mm off the long edge (Figure 9).

Figure 9, cutting and taping together the scale compartment.

The film is then bent into a U-shape that fits inside the scale backbone and taped in place along the tall sticks in the middle of the backbone. It may be necessary to cut the ends slightly to fit snugly in between the doors (Figure 10A).

Figure 10A

The edges of the film should not push against the door frames, but there should also not be gaps larger than 1-2 mm. A new piece of film is simple to install at this stage if the gaps are too large after cutting. When a good enough fit is achieved, the film is epoxy glued to the backbone along the U shaped parts hugging it. Enough epoxy should be applied to also firm up any loose push-fit junctions between the 3D-printed parts.

Next, the two 1 mm thick acrylic pieces prevent_burrowing.svg (part V), that prevent burrowing below the scale, are attached using a drop of chloroform to the door frame guides about 2 mm below the transparent film (Figure 10B).

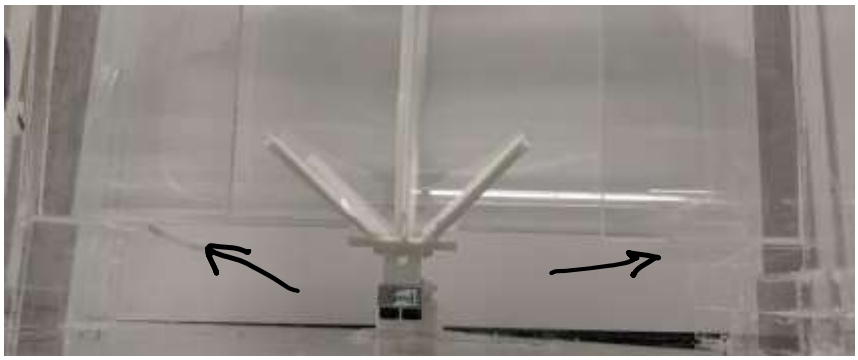


Figure 10B.

Then the four 3 mm thick acrylic pieces sem_support.svg (part R), to support the sides of the scale, are attached using drops of chloroform to the door frame guides, about 2 mm away from the transparent film (Figure 10C).

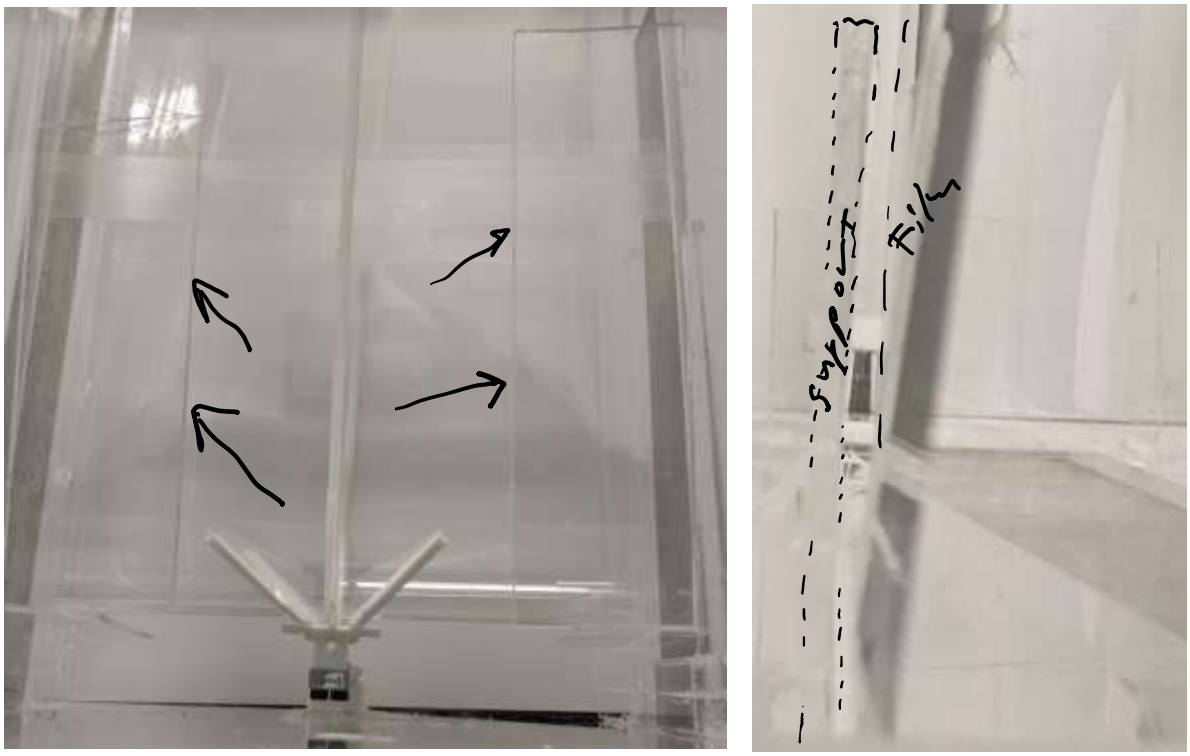
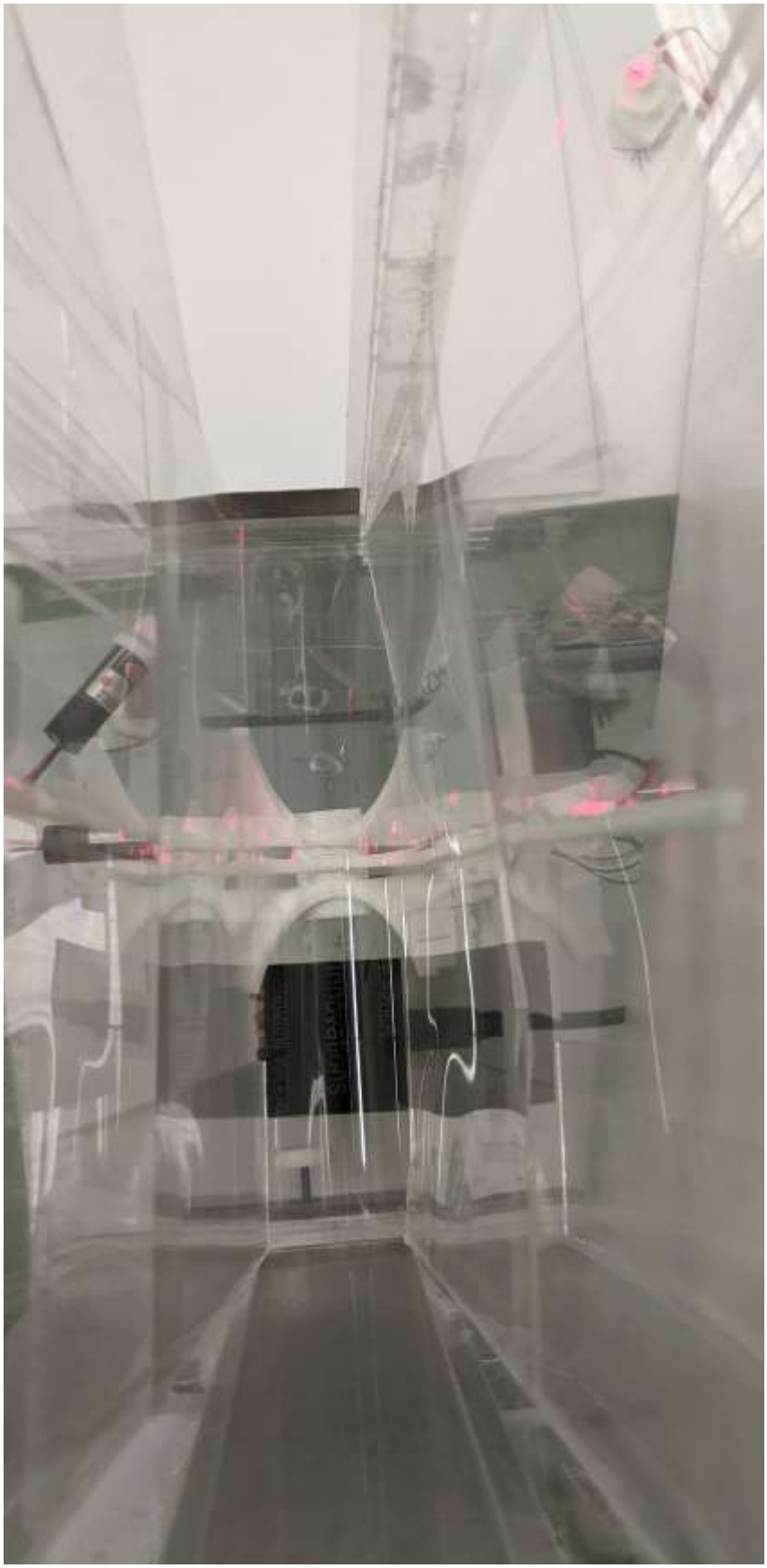


Figure 10C.

4) Install the beam break detectors:

Lastly, the beam breaks are installed using 3D-printed parts E-J, diode lasers (Sparkfun XXX) and photoresistors wired to resistors. The laser and detector holders are built first (each has 3 parts: base E, tightening nut F and one of parts G-J with a ball joint). The ball joint may need to be filed down depending on the grain of the print. After assembly, the parts are placed at their approximate locations (Figure 11), door 1 is set to protrude approximately 50 mm from the floor plate, and then beam targeting is checked by sighting through the laser holders to the detectors. Beam 1 needs to pass in the middle of the entrance about 10 mm above door 1. Beam 2 needs to pass about 10 mm above the bottom of the U-shaped film at a distance of >100 mm from door 2. A small drop of chloroform is used to weakly bond each holder to the floor plate, so they can be detached if necessary. Later when the lasers and doors are operational, locations can be confirmed and the holders should be welded strongly with additional drops of chloroform.



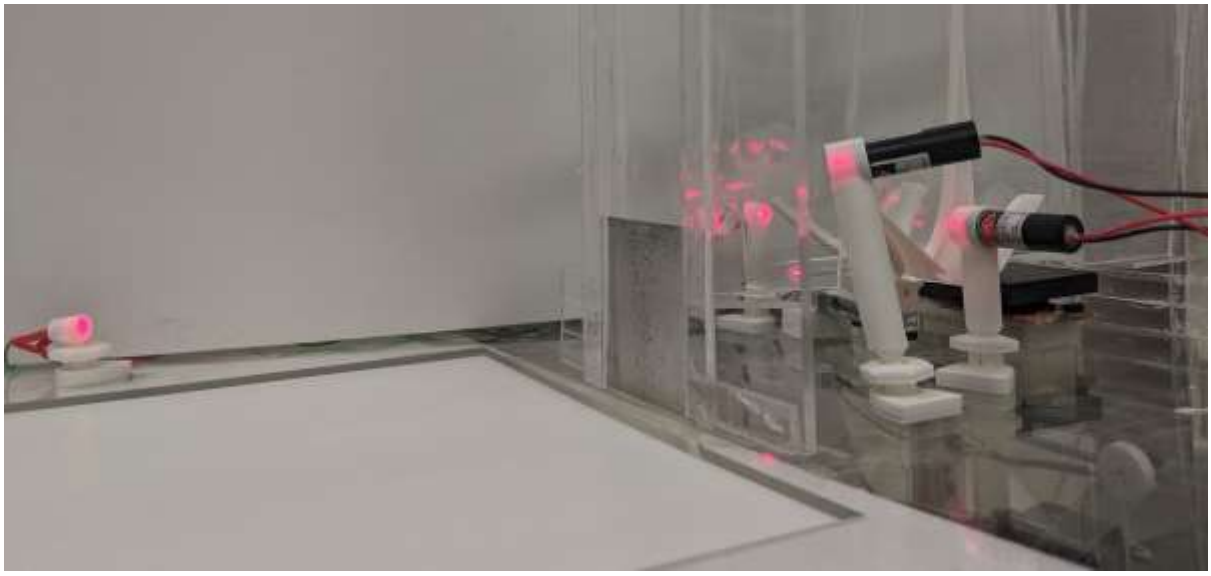
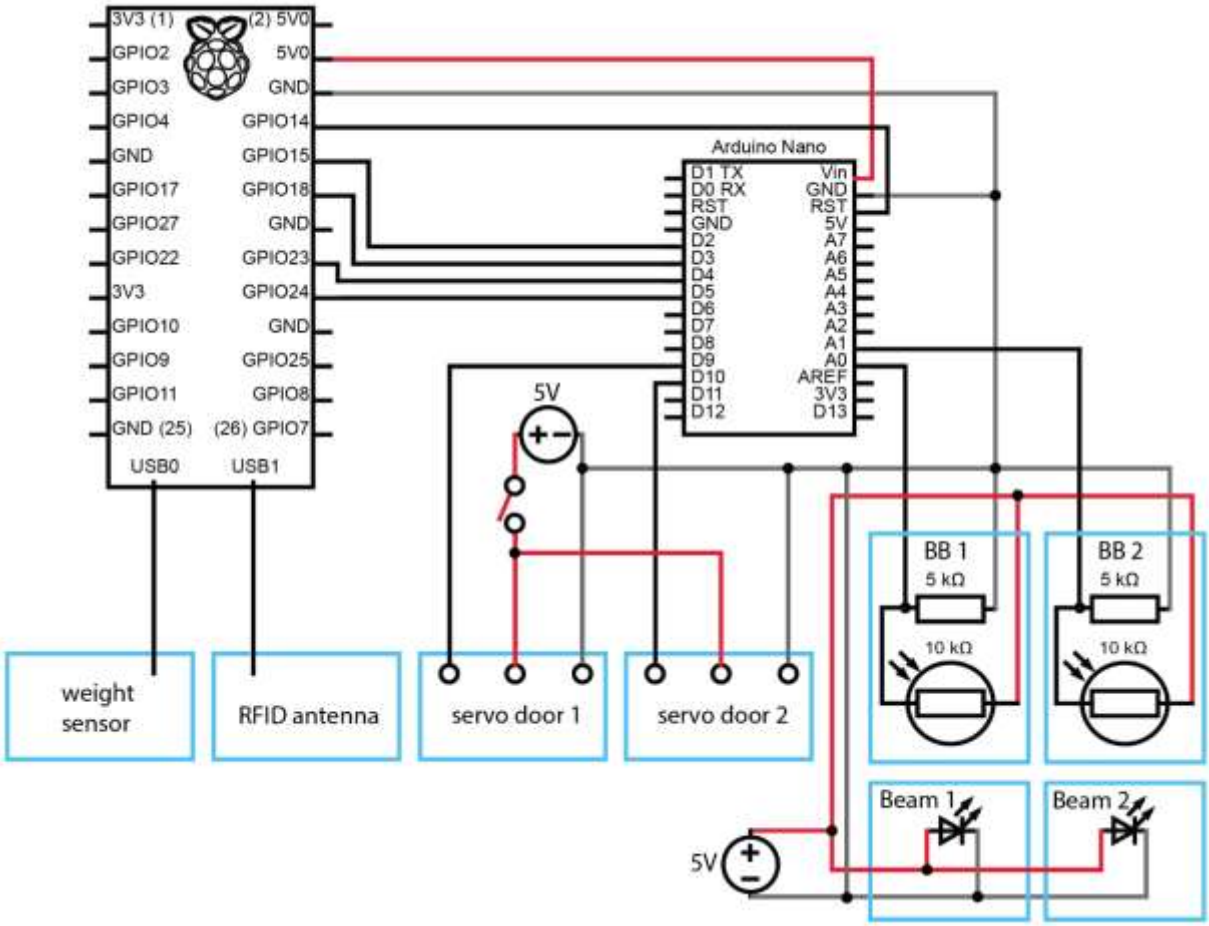


Figure 11.

Electronics



Code

Arduino

An Arduino board is used to detect beam breaks, reporting them to the Raspberry Pi, and open/close doors when commanded by the Raspberry Pi. The following code (MaheshKarnani, 2022a) is flashed to an Arduino Nano (also Uno and Mega boards were tested, making sure to use PWM pins for the servos):

```
#include<Servo.h>
Servo servo1;     // Servo 1 = door1
#include<Servo.h>
Servo servo2;     // Servo 2 = door2

//Constants
const int slowness = 8000;   //slowness factor, us wait between 1deg movements, change to set door speed
//beam breaks
const int pResistor1 = A0;//BB1 safety
const int pResistor2 = A1;//BB2 return

//servo control
const int servoPin1 = 9;// door servos 1-2
const int servoPin2 = 10;

//outputs to Pi
const int ard_pi_BB1 = 4;//reports BB1low to Pi
const int ard_pi_BB2 = 5;//reports BB2low to Pi

//door angles: determine empirically
const int CLOSE_DOOR1 = 179;//Angle of 179 degrees -> door is closed
const int OPEN_DOOR1 = 76;//Angle of 76 degrees -> door is opened
const int CLOSE_DOOR2 = 179;//Angle of 179 degrees -> door is closed
const int OPEN_DOOR2 = 45;//Angle of 45 degrees -> door is opened

//motor inputs from Pi
const int pi_ard_door1 = 2;//open door1
const int pi_ard_door2 = 3;//open door2

//Variables
int photo_value1;//Store value from photoresistor (0-1023)
int INIT_READ1;//Store initial value from photoresistor
int photo_value2;//Store value from photoresistor (0-1023)
int INIT_READ2;
int pos1_current = CLOSE_DOOR1; //initial position variables for servos
int pos1_target = CLOSE_DOOR1;
int pos2_current = CLOSE_DOOR2; //initial position variables for servos
int pos2_target = CLOSE_DOOR2;
long start = millis();


void setup()
```

```
{
//  Serial.begin(9600);//setup serial
  pinMode(pResistor1, INPUT);//Set photoResistor - A0 pin as an input
  pinMode(pResistor2, INPUT);
  pinMode(ard_pi_BB1, OUTPUT);//output reports to Pi
  pinMode(ard_pi_BB2, OUTPUT);
  pinMode(pi_ard_door1, INPUT);//input commands from Pi
  pinMode(pi_ard_door2, INPUT);

  servo1.attach(servoPin1);
  servo1.write(OPEN_DOOR1);
  delay(1500);
  INIT_READ1 = analogRead(pResistor1);//calibrate top of door detector when door1 open
  delay(200);
  servo1.write(CLOSE_DOOR1);
  delay(1000);
  servo2.attach(servoPin2);
  servo2.write(OPEN_DOOR2);
  delay(1000);
  servo2.write(CLOSE_DOOR2);
  delay(1000);

  digitalWrite(ard_pi_BB1, LOW);//communication to Pi
  digitalWrite(ard_pi_BB2, LOW);

  INIT_READ2 = analogRead(pResistor2);// calibrate exit beam-break

//  Serial.print("INIT_READ1 ");  //show beam break values for trouble shooting if serial monitor is on
//  Serial.println(INIT_READ1);
//  Serial.print("INIT_READ2 ");
//  Serial.println(INIT_READ2);
}

void loop()
{
    photo_value1 = analogRead(pResistor1); //read beam breaks
    photo_value2 = analogRead(pResistor2);

//    Serial.print("photo_value1 ");
//    Serial.println(photo_value1);
//    Serial.print("photo_value2 ");
//    Serial.println(photo_value2);

    //output BB detectors to Pi
    if (photo_value1<INIT_READ1*0.85) //BB1 safety - inverted here
    {
       digitalWrite(ard_pi_BB1, LOW);
    }
    else
    {
       digitalWrite(ard_pi_BB1, HIGH);
```

```
}

if (photo_value2<INIT_READ2*0.9) //BB2 exit
{
   digitalWrite(ard_pi_BB2, HIGH);
}
else
{
   digitalWrite(ard_pi_BB2, LOW);
}

//servo movement loops
if (pos1_current<pos1_target) //SERVO 1
{
   pos1_current=pos1_current+1;
   servo1.write(pos1_current);
   delayMicroseconds(slowness);
}
if (pos1_current>pos1_target)
{
   pos1_current=pos1_current-1;
   servo1.write(pos1_current);
   delayMicroseconds(slowness);
}

if (pos2_current<pos2_target) //SERVO 2
{
   pos2_current=pos2_current+1;
   servo2.write(pos2_current);
   delayMicroseconds(slowness);
}
if (pos2_current>pos2_target)
{
   pos2_current=pos2_current-1;
   servo2.write(pos2_current);
   delayMicroseconds(slowness);
}

//target commands
//SERVO 1
if ((digitalRead(pi_ard_door1) == HIGH))
{
   pos1_target=OPEN_DOOR1;
}
else
{
   pos1_target=CLOSE_DOOR1;
}
//SERVO 2
if ((digitalRead(pi_ard_door2) == HIGH))
{
```

```
    pos2_target=OPEN_DOOR2;
  }
  else
  {
    pos2_target=CLOSE_DOOR2;
  }

}//void loop end
```

Raspberry Pi

A Raspberry Pi 4b is set up to log relevant events in the SEM and control the entry logic. Python 3.7.3 and the python libraries specified in the requirements.txt file here (MaheshKarnani, 2022b) need to be installed. The helper script SEM_functions.py should be in the same folder as the main script SEM_only.py. Two manual setup operations must be done every time after restarting the Pi: 1) The PiGPIO library is launched as a daemon by typing 'sudo pigpiod' in the command prompt. 2) A USB connection is established to the OpenScale module by opening a serial monitor connection to it in Arduino IDE, confirming weight data is being read, then closing Arduino IDE so the established connection is available for the Python script. Then the following script is run (we use Thonny Python IDE):

```
# Single entry module from Switch_maze
from SEM_functions import *
"""
Execution loop for single entry module demo.
Opens, detects one animal in module and reads its RFID, if animal hasn't
exited in nest_timeout, closes safely, weighs animal, passes to other side,
waits minimum_entry_time, then allows return.

Changes typically not recommended here.
This is an example.
"""
while True:

    # reset
    if MODE == 1:
        pi.write(pi_ard_door1, 1)  # open SEM
        print("\nSEM open\n")
        for x in range(np.size(animal_list)):
            animaltag = animal_list[x]
            tick = pi.get_current_tick()
            save.append_event("*", "", "entry_available", animaltag, tick)
        MODE=2

    # wait for entry
    if MODE == 2:
        print("Scanning RFID tag in scale")
        print(datetime.datetime.now())

        while True:
```

```
            animaltag = RFID_readtag("RFID1")
            if animaltag:
                w = read_scale()
                if w<light:
                    print("too light ")
                    print(w)
                if w>heavy:
                    print("too heavy ")
                    print(w)
                if not int(round(time.time()))-
animal_timer[animal_list.index(animaltag)]> nest_timeout:
                    print("too soon after last visit, time to next ")
                    print(int(round(time.time()))-
animal_timer[animal_list.index(animaltag)]-nest_timeout)
                if not pi.read(ard_pi_BB1):
                    print("BB1 detected something - not safe to close door 1")
                if (
                    w > light
                    and w < heavy
                    and pi.read(ard_pi_BB1)
                    and int(round(time.time()))
                    - animal_timer[animal_list.index(animaltag)]
                    > nest_timeout
                ):
                    pi.write(pi_ard_door1, 0)  # close door1
                    MODE = 3
                    choice_flag = False
                    entry_flag = False
                    water_flag = True
                    break
            else:
                MODE = 2
                print("*", end=",")
                break

    # correct animal on scale
    if MODE == 3:
        print("\nweighing\n")
        # Weighing for entry
        flag_heavy = acquire_weight(animaltag)
        if flag_heavy:
            # Append data
            tick = pi.get_current_tick()
            save.append_event(
                "+", "", "ENTRY DENIED MULTIPLE ANIMALS", animaltag, tick
            )
            print("ENTRY DENIED MULTIPLE ANIMALS")
            MODE = 1
```

```
        if not flag_heavy:
            pi.write(pi_ard_door2, 1)  # open door 2
            tick = pi.get_current_tick()
            save.append_event("", "", "entry", animaltag, tick)
            print("\nPASS\n")
            print(animaltag)
            print(datetime.datetime.now())
            time.sleep(minimum_entry_time)
            print("waiting for exit")
            print(datetime.datetime.now())
            MODE = 4

    # wait for exit
    if MODE == 4:
        if pi.read(ard_pi_BB2):
            pi.write(pi_ard_door2, 0)  # close door 2
            animal_timer[animal_list.index(animaltag)] =
int(round(time.time()))
            MODE = 1
```

The helper script SEM_functions.py is shown below, where the user can change parameters like how long the mouse must minimally spend on the other side before permitted through, heavy and light weight limits and where data is stored.

```
# Switch_maze SEM
import serial
import time
import pigpio
import os
import pandas as pd
import statistics as stats
import datetime
import numpy as np
"""
Functions and parameters for switch maze standard operation.
Change e.g., cohort tags and water amount here.
"""
#initialize ard during user input
pi = pigpio.pi() # init pigpio
pi_ard_door1 = 15
pi_ard_door2 = 18
pi.set_mode(pi_ard_door1, pigpio.OUTPUT)
pi.set_mode(pi_ard_door2, pigpio.OUTPUT)
ard_pi_BB1 = 23# reports BB1low
ard_pi_BB2 = 24# reports BB2low
pi.set_mode(ard_pi_BB1, pigpio.INPUT)
pi.set_mode(ard_pi_BB2, pigpio.INPUT)
```

```
pi.write(pi_ard_door1, 0)  # close door1
pi.write(pi_ard_door2, 0)  # close door2
PiArd_reset = 14
pi.set_mode(PiArd_reset, pigpio.OUTPUT)
pi.write(PiArd_reset, 0) # ard resets when reset pin is LOW
time.sleep(0.1)
pi.write(PiArd_reset, 1) # back HIGH so it stops resetting

# query parameter input from user
print('********** getting user input **********')
test_answer=int(input("test (0) or exp (1) ?"))
if test_answer==0:
    test_tags = ["202100030","137575399426", "2006010085"]#insert your test
tags here manually
    animal_list = test_tags
    print('TEST session with your usual test tags')
elif test_answer==1:
    animal_list = [
    "34443624695",
    "34443624808",
    "137575399507",
    "34443624982",
    ]  # insert your mouse tags here manually
    print('LIVE session with subjects:')
    print(animal_list)
print('********************** input complete, starting session
**********************')

# hard-coded recording parameters
minimum_entry_time = 6 # seconds to wait until starting to detect exit
nest_timeout = 10  # timeout in nest after exiting maze in seconds
heavy = 35 #g limit too heavy = more than one animal on scale
light = 10 #g limit too light = animal incompletely on scale
chuck_lines = 2  # chuck first weight reads for stability

# document data folder
os.chdir("/home/pi/Documents/Data/")

# initialize serial port for usb RFID
serRFID = serial.Serial()
serRFID.port = "/dev/ttyUSB1" # user may need to change this
serRFID.baudrate = 9600
serRFID.timeout = 100  # timeout in seconds when using readline()
serRFID.open()
if serRFID.is_open == True:
    print("\nRFID antenna ok. Configuration:\n")
    print(serRFID, "\n")  # print serial parameters
serRFID.close()
```

```
# initialize serial port for usb OpenScale
ser = serial.Serial()
ser.port = "/dev/ttyUSB0" # user may need to change this
ser.baudrate = 19200
ser.timeout = 100
# specify timeout when using readline()
ser.open()
ser.flush()
for x in range(10):
    line = ser.readline()
    print(line)
if ser.is_open == True:
    print("\nScale ok. Configuration:\n")
    print(ser, "\n")  # print serial parameters

# animal timers
animal_timer = animal_list.copy()
for x in range(np.size(animal_list)):
    animal_timer[x] = int(round(time.time())) - nest_timeout


# initialize state variables
MODE = 1

# functions
def RFID_readtag(RFIDnum):
    """
    This function reads the RFID tag, removes junk and returns the
    converted ID from hexadecimal to decimal.
    """
    RFIDtimer = int(round(time.time()))
    if RFIDnum == "RFID1":
        try:
            serRFID.close()
            serRFID.open()
            serRFID.flush()
            junk = serRFID.read(1)
            tag = serRFID.read(10)
            checksum = serRFID.read(2)
            junk2 = serRFID.read(3)
            animaltag = str(int(tag, 16))  # transform from hexadecimal to a
number
            print(animaltag)
            serRFID.close()
            return animaltag
        except:
            serRFID.close()
            print("RFID read failed")
```

```
            animaltag = False
            return animaltag

def acquire_weight(animaltag):
    global chuck_lines
    """
    This function acquires 50 datapoints of the animal's weight and returns
    mean, median, mode, max_mode(the latter does not
    work in python 3.7).
    """
    print("Acquiring weight")
    flag_heavy = False
    ys = []  # store weights here
    ser.close()
    ser.open()
    ser.flush()
    for x in range(chuck_lines):  # chuck lines
        line = ser.readline()
    for x in range(50):  # 50 lines*120ms per line=6s of data
        line = ser.readline()
        if x % 1 == 0:
            print(line)
        line_as_list = line.split(b",")
        relProb = line_as_list[0]
        relProb_as_list = relProb.split(b"\n")
        relProb_float = float(relProb_as_list[0])
        relProb_float = relProb_float * 1000
        # More than one animal:
        if relProb_float > heavy:
            print("MULTIPLE ANIMALS ON SCALE")
            flag_heavy = True
            return flag_heavy
        else:
            ys.append(relProb_float)

    if not flag_heavy:
        for i in range(len(ys)):
            ys[i] = round(ys[i], 3)
        weight_data_mean = stats.mean(ys)
        weight_data_median = stats.median(ys)
        # mode
        try:
            weight_data_mode = stats.mode(ys)
        except:
            weight_data_mode = "NO MODE"
        # mode max
        try:
            weight_data_max_mode = stats.multimode(ys)
```

```
            weight_data_max_mode = weight_data_max_mode[-1]  # largest of
modes
        except:
            weight_data_max_mode = "NO MAX_MODE"
        # appending data to database
        save = SaveData()
        save.append_weight(
            weight_data_mean,
            weight_data_median,
            weight_data_mode,
            weight_data_max_mode,
            animaltag,
        )
        return flag_heavy

def read_scale():
    """
    This function takes a quick read to sense the scale.
    """
    global chuck_lines
    m = []  # store weights here
    ser.close()
    ser.open()
    ser.flush()
    for x in range(chuck_lines):  # chuck lines
        line = ser.readline()
    for x in range(2):  # 2 lines*120ms per line=0.24s of data
        line = ser.readline()
        line_as_list = line.split(b",")
        relProb = line_as_list[0]
        relProb_as_list = relProb.split(b"\n")
        n = float(relProb_as_list[0])
        n = n * 1000
        m.append(n)
    w = stats.mean(m)
    return w

class SaveData:
    def append_weight(
        self,
        weight_data_mean,
        weight_data_median,
        weight_data_mode,
        weight_data_max_mode,
        animaltag,):
        """
        Function used to save weight data to a .csv file
        """
```

```
        weight_list = {
            "Weight_Mean": [],
            "Weight_Median": [],
            "Weight_Mode": [],
            "Weight_Max_Mode": [],
            "Date_Time": [],
        }
        weight_list.update({"Weight_Mean": [weight_data_mean]})
        weight_list.update({"Weight_Median": [weight_data_median]})
        weight_list.update({"Weight_Mode": [weight_data_mode]})
        weight_list.update({"Weight_Max_Mode": [weight_data_max_mode]})
        weight_list.update({"Date_Time": [datetime.datetime.now()]})
        df_w = pd.DataFrame(weight_list)
        print(df_w)
        if not os.path.isfile(animaltag + "_weight.csv"):
            df_w.to_csv(animaltag + "_weight.csv", encoding="utf-8-sig",
index=False)
            print("weight file created")
        else:
            df_w.to_csv(
                animaltag + "_weight.csv",
                mode="a+",
                header=False,
                encoding="utf-8-sig",
                index=False,
            )
            print("weight file appended")

    def append_event(self, rotation, food_time, event_type, animaltag,
hardware_time):
        """
        Function used to save event data to a .csv file
        """
        global event_list
        event_list = {
            "Date_Time": [],
            "Rotation": [],
            "Pellet_Retrieval": [],
            "Type": [],
            "hardware_time": [],
        }
        event_list.update({"Rotation": [rotation]})
        event_list.update({"Pellet_Retrieval": [food_time]})
        event_list.update({"Type": [event_type]})
        event_list.update({"Date_Time": [datetime.datetime.now()]})
        event_list.update({"hardware_time": [hardware_time]})
        df_e = pd.DataFrame(event_list)
        if not os.path.isfile(animaltag + "_events.csv"):
```

```
            df_e.to_csv(animaltag + "_events.csv", encoding="utf-8-sig",
index=False)
        else:
            df_e.to_csv(
                animaltag + "_events.csv",
                mode="a+",
                header=False,
                encoding="utf-8-sig",
                index=False,
            )

#final init calls
save = SaveData()

# save session parameters in a begin session event line
for x in range(np.size(animal_list)):
    animaltag = animal_list[x]
    tick = pi.get_current_tick()
    save.append_event(
        minimum_entry_time, nest_timeout, "begin session", animaltag, tick
    )
```

Adjustment

Beam targeting

Beam break 1 is used for safely closing door 1: Beam 1 is targeted through door 1 such that when the door is open the beam hits the photoresistor. When the door is closed or an animal is on top of the open door, the beam is broken. Beam break 2 is used for detecting an exiting animal at a safe distance from door 2 (>100 mm): Beam 2 is targeted through the sides of the single entry module such that an animal in the middle of the module will break it.

Beam detector readings can be monitored in Arduino IDE serial monitor after uncommenting lines 44, 70-73 and 81-84 in the Arduino code. After diagnostics, these lines need to be commented out again to operate doors rapidly. Beam detection thresholds are set on lines 87 and 96.

Weight sensor parameter selection and calibration

The weight sensor is set up and calibrated according to the manufacturer's instructions https://learn.sparkfun.com/tutorials/openscale-applications-and-hookup-guide?_ga=2.14390071.1017329722.1674997605-1611569064.1667393722. Briefly, a serial connection to the OpenScale is launched in Arduino IDE and the control menu is accessed by pressing 'x'. Baudrate is set to 19200, report rate 120 ms, units kg, decimals 4, average amount 1, serial trigger 'off'. The scale is then tared to zero and calibrated with a 25 g weight. Calibration should be repeated daily in the beginning as the load cell can 'creep' somewhat after installation.

Door angles and speed

The angles that the servo needs to achieve to close and open the door will likely vary based on the exact distances used in each build, and are set by trial and error on lines 21-24 of the Arduino code. As a standard practice, install the levers such that they cannot hit the floor board above them, i.e., power off the servo, turn the servo hub to the extreme position and install the lever there such that it is near but not touching the floor. Because we do this, the default code has ‘closed’ values of 179 degrees. After that it will be simple to find a suitable ‘open’ value by testing values less than 90 degrees. In case the door lever collides with something, turn the power to the servos off rapidly from their power switch, go back to the code and bring the degree value closer to 90. When powering up the system the servos can receive extreme commands. Therefore it is best to power up the servos last.

The speed of door movements is set by the slowness constant on line 7 of the Arduino code. This value corresponds to the time, in microseconds, it takes to move the servo by 1 degree. We have found 8000 ideal.