Supplementary Information

Pellet dispensomixer and pellet distributor: Open hardware for nanocomposite space exploration via automated material compounding

Authors

Miguel Hernández-del-Valle,[a,b] *Jorge Ilarraza-Zuazo,*[a,b] *Enrique Dios-Lázaro,*[a,b] *Javier Rubio,*[a] *Joris Audoux,*[a,c] *and Maciej Haranczyk*[*a]

Affiliations

[a] *IMDEA Materials Institute, c/Eric Kandel, 2, 28906 Getafe, Madrid, Spain.*

[b] *Universidad Carlos III de Madrid, 28911 Leganés, Madrid, Spain.*

[c] *Université de Limoges, 87032 Limoges, France.*

Corresponding author's email address

maciej.haranczyk@imdea.org

1. Pellet dispensomixer

1.1 Design files summary

Design filename	Number of copies	Number of copies
StructureBase	2	Base that stores the different electronic components of the dispenser, including motor connections, cables, resistors and the Arduino board
StructureTop	2	Covers the StructureBase, protecting the connections and the plate, in addition this part is attached to the DispenserBase with holes for a better position with respect to the electronics
StructureUnion	8	Piece that serves as a union between the two halves of the StructureBase, providing support and rigidity
DispenserBase	8	Support for each of the dispensers
DispenserTop	8	Part that regulates the output of the pellets from the dispenser
RotorSmall / RotorLarge / RotorXL	8*	The rotors are the only moving parts of the entire dispenser. It is available in three designs with different hole sizes, so for each dispenser it can be chosen depending on the sizes of the pellets that it is goint to contain
Funnel	8	helps the filling of the dispensers
IonizerSupport	1	Part designed to include a ionizer to eliminate static charge in the pellets that can delay the dispensing process. The airflow of a single ionizer is distributed towards the eight dispensers.

1.2 Bill of materials

Item	Quantity	Unit Price (€)	Link
PLA Filament spool	2	20,61	https://tienda.sicnova3d.com/consumibles-sicnova/13041-4090-filamento-para-impresoras-3d-pla-sicnova-750gr-285mm#/170-diametro_bobina_sicnova-bobina_285_mm/175-color_sicnova-azul
Arduino Mega 2560 Rev 3	1	49,55	https://es.rs-online.com/web/p/arduino/7154084?gb=s
Breadboards	3	5.57	https://es.rs-online.com/web/p/placas-de-prueba/1892277
24BYJ48 stepper motor	8	6.23	https://es.rs-online.com/web/p/kits-de-desarrollo-de-alimentacion-motores-y-robots/1793741
A4988 driver	8	2.84	https://www.electronicaembajadores.com/es/Productos/Detalle/LCMM038/modulos-electronicos/drivers-de-motor/controlador-motor-paso-a-paso-con-a4988-2-0-a/
$100\mu F$ capacitor	8	0.082	https://es.rs-online.com/web/p/condensadores-de-aluminio/7111396
Ball bearing (Ø int. 8mm, Ø ext. 24mm, width 8mm)	8	4.31	https://es.rs-online.com/web/p/rodamientos-de-bola/2078240
Wall power source 12V	1	13.82	https://es.rs-online.com/web/p/adaptadores-ac-dc/1391779
Ionizer	1	571.87	https://es.rs-online.com/web/p/ionizadores/1260265?gb=s
Transparent PVC pipe (Ø int. 40mm, Ø ext. 50mm)	8x15cm	12*	-
Cables and board bridges	Assorted box	10*	-
M6x25mm screws with M6 nuts	4	0.31*	-
M5x16mm screws with M5 nuts	16	0.25*	-
M4x10mm screws with M4 nuts	16	0.17*	-
M3x25mm screws with M3 nuts	32	0.12*	-

*Estimated prices

1.3 **Build instructions**

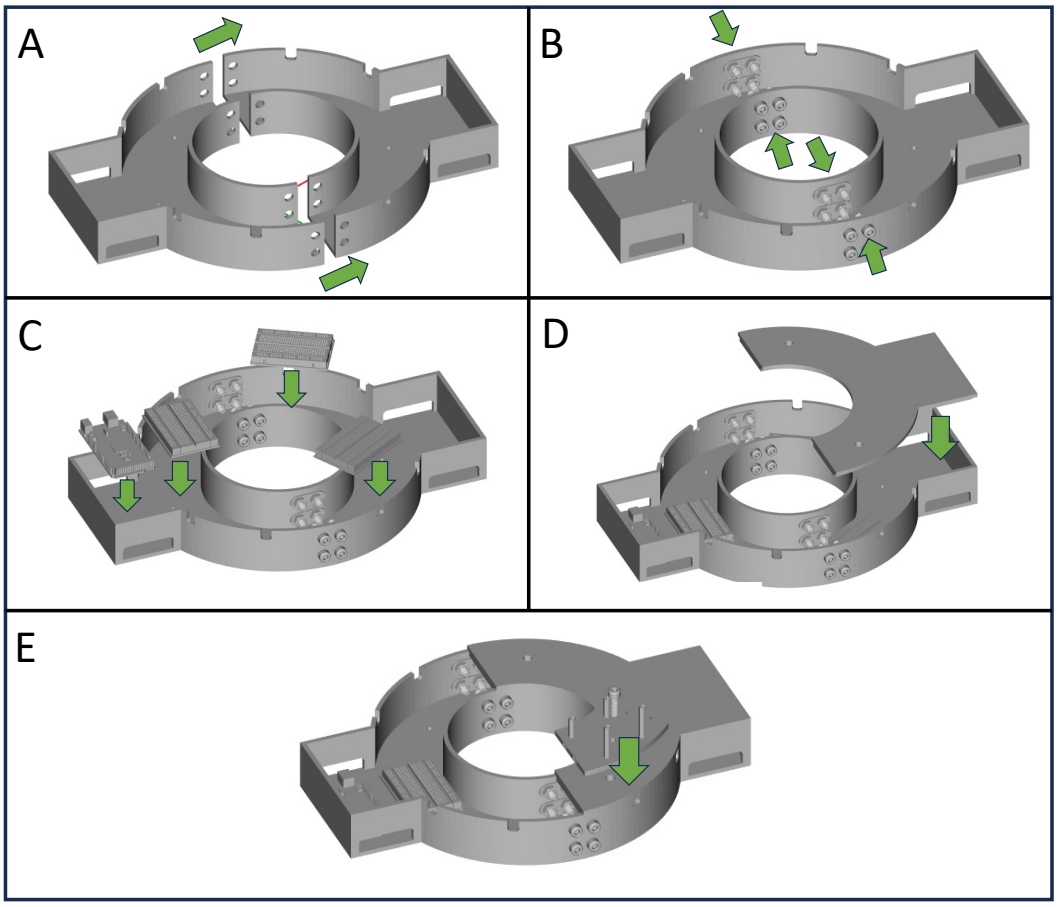

Figure S1: First steps for building the dispensomixer. Two copies of StructureBase are brought together (A) and secured using eight copies of StructureUnion, each of them joint with two M5 screws and nuts (B). Then, the breadboards are distributed along the circumference and the Arduino Mega board is placed in the cavity(C). The circuit should then be assembled, as described in subsection 1.4. After that, StructureTop is placed (D) to protect the circuit. Lastly, DispenserBase is placed (E) and secured with M6 screw and nut.

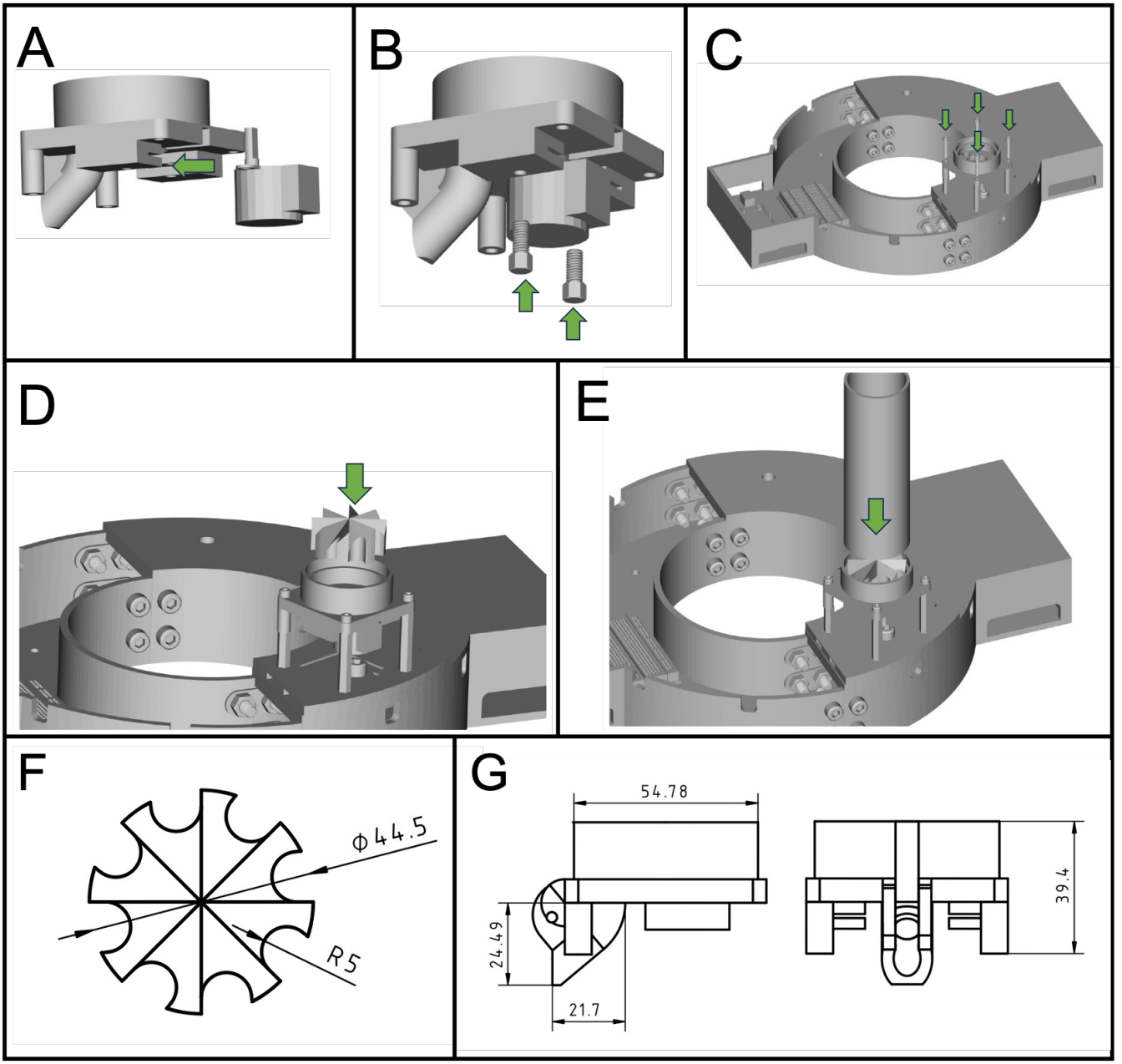

Figure S2: Steps to be repeated for each of the eight dispensers: Introduce one of the 24BYJ-48 stepper motors in the slits of DispenserTop (A) and secure with two M4 screws (B). The DispenserTop is placed and fixed on top of the DispenserBase using four M3 screws (C). Next, a ball bearing is attached to the rotor shaft and the rotor is introduced (D), aligned with the motor shaft. There are three available models for the rotor, depending on the size of the pellets. Finally, we add a transparent PVC pipe to serve as deposit (E). F and G show the schematics of the rotor and base, respectively, to help get an idea of the dimensions.

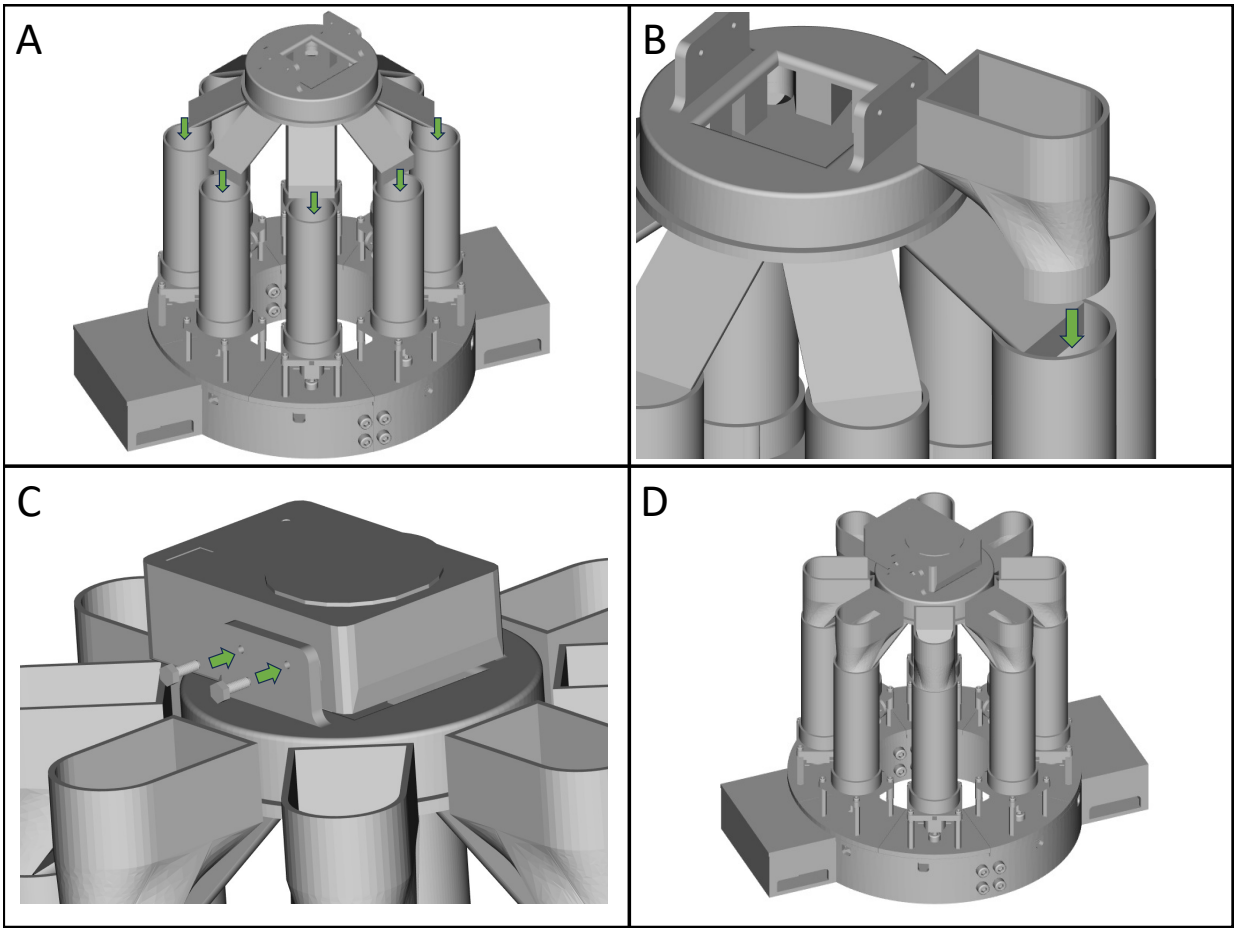

Figure S3: Steps to place the ionizer: IonizerSupport is placed so it can distribute the air to the eight dispensers (A). Then, Funnels are placed on top of each dispenser to facilitate the material filling (B). Ionizer can then be placed and secured with M3 screws (C). The dispenser is now ready to be operated (D).

1.4 Circuit assembly

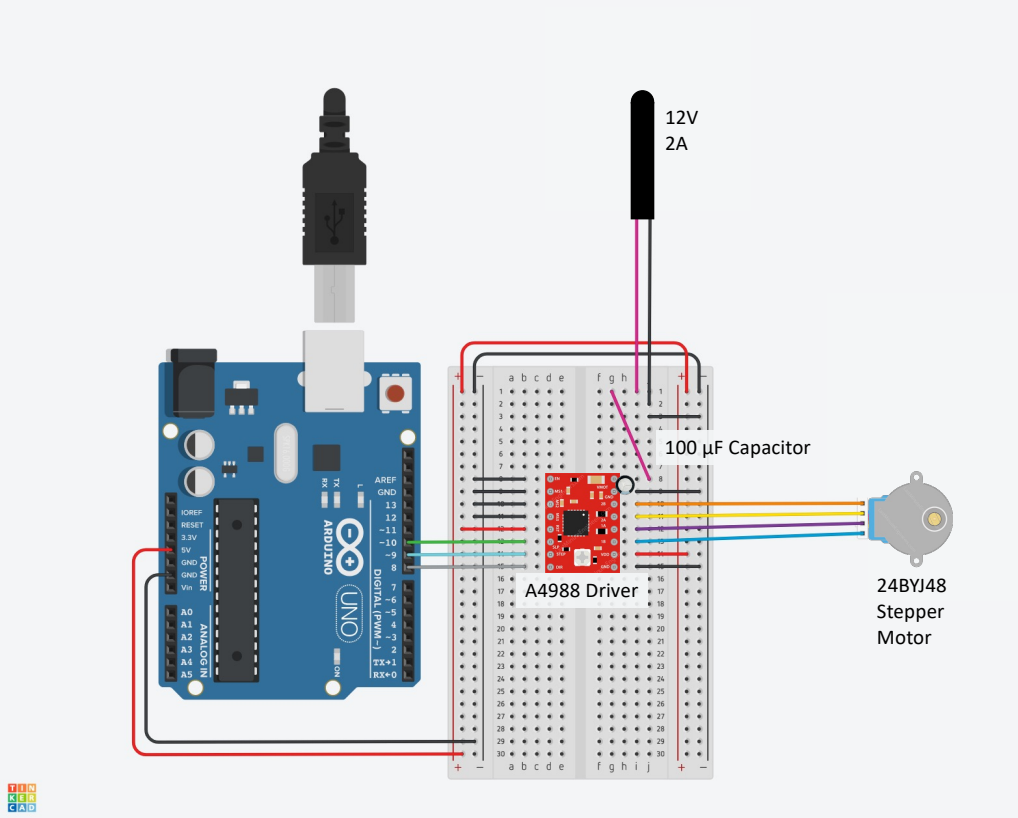

Figure S4: Schematics of the circuit for each of the motors. This circuit should be repeated for as many motors as needed, so in our case we use Arduino MEGA instead of UNO to have more pins. The external power source provides the $> 8V$ needed for the A4988 driver, in this case using a 12V wall power source. It's essential that it has common ground with the Arduino. It is necessary to adjust the current in the driver by using the screw and a multimeter† . Warning: don't power the circuit before ensuring that everything is properly connected, and don't modify connections if the circuit is powered.

1.5 Operation instructions

1.5.1 Example using pellet dispenser

This notebook contains an example of how to run the code for using the dispensomixer. Most cells can be run without any modification if the dispenser has been built without design modifications.

```
# Import Libraries
import serial
import numpy as np
import time
import serial.tools.list_ports
import os
import sys
sys.path.append('src')
from DispenserSetup import SetupDispenser, Compositions
```

[0]† https://ardufocus.com/howto/a4988-motor-current-tuning/

In this cell, we define the port name for the Arduino and the balance. We do it by listing the connected ports and selecting the ones whose device serial is the corresponding to our device. You should find the serial of your devices and change it accordingly. You can do it by inspecting the devices detected by `serial.tools.list_ports.comports()`. In case of doubt, try disconnecting and connecting to see which ports are affected.

```
ports = serial.tools.list_ports.comports()

##############################################################################

arduino_serial = "write_here_your_arduino_serial"
balance_serial = "write_here_your_balance_serial"

##############################################################################

for port in ports:
    if port.serial_number == arduino_serial:
        arduino_port = port.device
    elif port.serial_number == balance_serial:
        balance_port = port.device

### Serial initialization
## Arduino
arduino = serial.Serial(arduino_port, 9600)
## Balance
balance = serial.Serial(port=balance_port, baudrate=9600, bytesize=serial.SEVENBITS,
                        parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
```

In the next cell, we use the Arduino Command Line Interface (arduino-cli) to compile and upload the arduino script. If you prefer to do it using the Graphical User Interface, skip this cell. For using it, update the location of your arduino-cli

```
##############################################################################

os.system("PATH/arduino-cli compile --fqbn arduino:avr:mega 8noIRnew_NOshaketimer_new/8
    noIRnew_NOshaketimer_new.ino")
os.system("PATH/arduino-cli upload -p "+arduino_port+" --fqbn arduino:avr:mega 8
    noIRnew_NOshaketimer_new/8noIRnew_NOshaketimer_new.ino")

##############################################################################
```

The next step is to define the materials and their concentrations in each of the dispensers. For that we will use the `SetupDispenser` function from `/src/DispenserSetup`. It will require a `.csv` file detailing the compositions, that can be built following the example available in the help of the function:

```
help(SetupDispenser)
```

```
##############################################################################
D, M, A = SetupDispenser("data/dispenser_setup.csv", "PLA")
##############################################################################
```

Next, we read the file that contains the compositions using the Compositions function. It's working is detailed in the documentation:

```
help(Compositions)
```

```
##############################################################################
compositions = Compositions("./data/compositions_to_dispense.csv", D, M, A, "PLA")
##############################################################################
print(compositions)
```

For the rest of the notebook we will use only the first of the compositions available in the .csv file. in your application you can easily loop over all of them, just remember to change the cup in the balance between compositions.

```
C = compositions[0]
```

It is convenient now to establish a ratio between materials: in this way, instead of dispensing all the amount of one material and then of another, we can alternate small quantities and thus have a better mixing. We do this by selecting the one with lower amount and expressing the others as a ratio with respect to this one. In every step we will dispense 1 g of the smaller one and the corresponding grams of the rest.

```
tol = 1e-3
minC = np.min([c for c in C if c>tol])
ratios = C/minC
print('The ratios for pellet dispensing will be: ', ratios)
```

The next cell establish commands to tare, calibrate and measure weight from the balance. You may need to edit them if your balance uses a different protocol.

```
#############################################################################
def tare(balance):
    # Set the balance measurement to 0.0 g
    try:
        balance.write(b'T\r\n')
    except:
        balance = serial.Serial(port=balance_port, baudrate=9600, bytesize=serial.SEVENBITS,
                        parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
        balance.write(b'T\r\n')


def calib(balance):
    # Calibrate the balance automatically
    try:
        balance.write(b'C\r\n')
    except:
        balance = serial.Serial(port=balance_port, baudrate=9600, bytesize=serial.SEVENBITS,
                        parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
        balance.write(b'C\r\n')

def measure(balance):
    try:
        trash = balance.read_all()
        balance.write(b'B\r\n')
        reading = balance.readline()
        weight = float(reading[:10])
    except:
        balance = serial.Serial(port=balance_port, baudrate=9600, bytesize=serial.SEVENBITS,
                        parity=serial.PARITY_EVEN, stopbits=serial.STOPBITS_ONE)
        trash = balance.read_all()
        balance.write(b'B\r\n')
        reading = balance.readline()
        weight = float(reading[:10])
    return weight
#############################################################################
```

Select the total amount of material that you want to dispense

```
#############################################################################
mass_target = 100 # g
#############################################################################
```

Now, we define some variables and arrays that we will need in the dispensing loop

```
measureAll = True
massAll = 0
mass = 0
masses = [[0] for _ in range(8)]
i = 0
times = [[0] for _ in range(8)]

```

```
dispenser_switch = {0 : b'0', 1 : b'1', 2 : b'2', 3 : b'3',
                    4 : b'4', 5 : b'5', 6 : b'6', 7 : b'7', 'stop' : b'8'}
```

The next cell runs the dispensers until the desired quantity is reached. As pointed above, it will alternate dispensers keeping the ratios between materials.

```
start = time.time()

while measureAll == True:
    for j in range(len(C)):
        if C[j] > tol:
            if masses[j][i] < ratios[j]*(i+1):
                measureNow = True
                tare(balance)
                time.sleep(2)
                arduino.write(dispenser_switch[j])
                while measureNow == True:
                    mass = measure(balance)
                    if mass + masses[j][i] >= ratios[j]*(i+1):
                        arduino.write(dispenser_switch['stop'])
                        time.sleep(4)
                        mass = measure(balance)
                        measureNow = False
                massAll = massAll + mass
                masses[j].append(masses[j][i]+mass)
                times[j].append(time.time()-start)
                time.sleep(0.1)
            elif masses[j][i] >= ratios[j]*(i+1):
                arduino.write(dispenser_switch['stop'])
            print("Dispenser ", j, "| iteration: ", i+1, "| t = ", times[j][i+1], "s | ", "
    mass : ", masses[j][i+1])
    if massAll > mass_target:
        measureAll = False
        arduino.write(dispenser_switch['stop'])
    i = i + 1

arduino.write(dispenser_switch['stop'])
```

We can now plot the dispensing progress, comparing the amount that was actually dispensed and the one asked by the code

```
import matplotlib.pyplot as plt

colors = ["C0", "C1", "C2", "C3", "C4", "C5", "C6", "C7"]

for key in D.keys():
    plt.plot(times[D[key]-1], masses[D[key]-1],label=key, c=colors[D[key]-1])
    plt.plot(times[D[key]-1], np.arange(len(times[D[key]-1]))*ratios[D[key]-1], '--',c=colors
    [D[key]-1])
plt.plot(0,0,'--',c = 'gray',label = "Expected amounts")
plt.title("Progression of Sample")
plt.xlabel("Time [s]")
plt.ylabel("Mass [g]")
plt.legend()
plt.show()
```

Finally, we present a measure for accuracy consistent of the total error committed divided by the total amount dispensed:

```
final_masses = [mass[-1] for mass in masses[:]]
accuracy = (1-np.sum(abs(final_masses-C*mass_target))/np.sum(final_masses))*100
print("Accuracy: ", accuracy, "%")
```

1.6 Estimation of the Workflow Acceleration

For a sample composition of PLA with 1% MMT + 1% CLO, the dispenser achieved an average mass flow rate of 7.57 g/min. The same task performed by a human, who usually can't dedicate more than 3 hours a day to composition preparation, recorded a mean MFR of 3.75 g/min. The acceleration factor is, therefore, estimated as it follows, with the result indicating an automated process which is more than twice as fast as the traditional.

$$AF = \left(\frac{MFR_{auto}}{MFR_{manual}} - 1 \right) \cdot 100 \approx 101.2 \tag{1}$$

In terms of time investment, the logistics of switching from a traditional to automated method must be considered as well. For this reason, an estimate of the amortization period was also calculated. With the instructions provided, a period of 1 day is enough to assemble the device, an initial time investment which, thanks to the acceleration factor, pays off in less than two days as shown in the Figure 1.6 A. This gain in time resources usage is also indicated as a Sankey diagram (Figure 1.6 B), where transition from traditional to automated procedure saves up 36.7% of the time. This time can then be allocated to more important tasks which do require an operator.

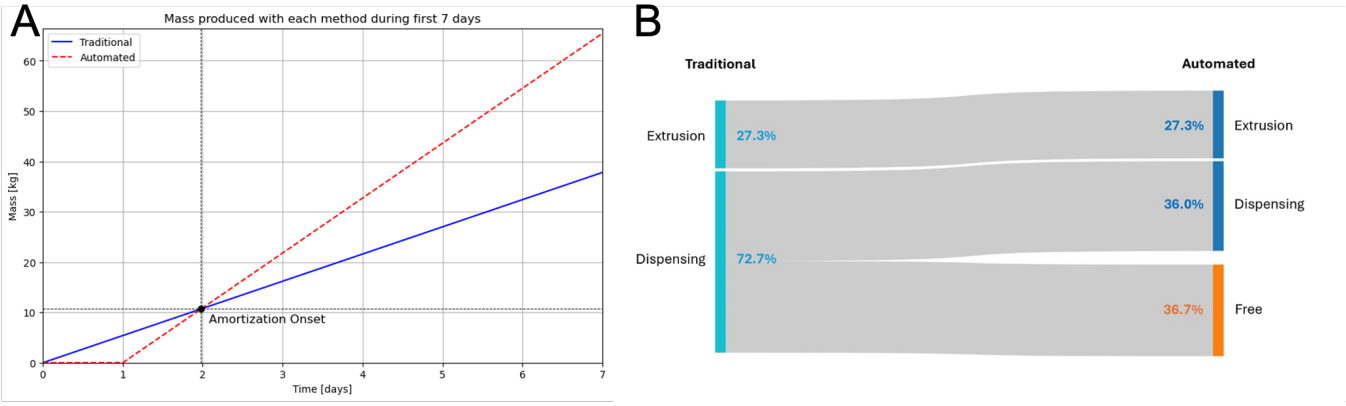

Figure S5: Comparison of output from dispensing manually or by using the automated dispenser. A: Mass produced by each method. The automated version includes in the calculation the assembly time, so it is possible to estimate that the amortization takes place in less than two days. B: Sankey diagram showing how the automated procedure can save around 36.7% of production time.

2. Pellet distributor

2.1 Design files summary

Design filename	Number of copies
HalfClamp	4
NemaMount	2
TopFunnel	1
IdlerLink	1
1mmPinion	1
1mmIdler	1
IdlerCap	1
Cover	1
CircuitCase	1
Elbow1	1
Elbow2	1
Elbow3	1
ValvePinion	1
JointPinion	1
Handle	1
Gear	1
BallStop	1
Shaft	1
MotorLink	1
ValveHousing	1
Pin	1
Disc	1
Nozzle	1
Tray	1

2.2 Bill of materials

Item	Quantity	Unit Price (€)	Link
PLA Filament spool	2	20.61	https://tienda.sicnova3d.com/consumibles-sicnova/13041-4090-filamento-para-impresoras-3d-pla-sicnova-750gr-285mm#/170-diametro_bobina_sicnova-bobina_285_mm/175-color_sicnova-azul
Arduino UNO 2560 Rev 3	1	28.98	https://es.rs-online.com/web/p/arduino/7154081?gb=s
Breadboards	1	5.57	https://es.rs-online.com/web/p/placas-de-prueba/1892277
24BYJ48 stepper motor	2	6.23	https://es.rs-online.com/web/p/kits-de-desarrollo-de-alimentacion-motores-y-robots/1793741
Nema17 stepper motor 12V 200 steps MMPP04	1	12.09	https://www.electronicaembajadores.com/es/Productos/Detalle/MMPP004/motores-servomotores-actuadores-lineales/motores-paso-a-paso/nema-17-motor-paso-a-paso-12-vcc-angulo-1-8-200-pasos-0-45n-m-42bygh40-1704a/
A4988 driver	3	2.84	https://www.electronicaembajadores.com/es/Productos/Detalle/LCMM038/modulos-electronicos/drivers-de-motor/controlador-motor-paso-a-paso-con-a4988-2-0-a/
$100\mu F$ capacitor	3	0.082	https://es.rs-online.com/web/p/condensadores-de-aluminio/7111396
Wall power source 12V	1	13.82	https://es.rs-online.com/web/p/adaptadores-ac-dc/1391779
Ionizer	1	571.87	https://es.rs-online.com/web/p/ionizadores/1260265?gb=s
Balance	1	284.63	https://es.rs-online.com/web/p/balanzas/1231737
Ball bearing (Ø int. 8mm, Ø ext. 24mm, width 8mm)	2	4.31	https://es.rs-online.com/web/p/rodamientos-de-bola/2078240
PVC pipe (Ø int. 45mm, Ø ext. 50mm)	10cm + 6cm	10*	-
Cables and board bridges	Assorted box	10*	-
M3x10mm screws with M3 nuts	10	0.12*	-
M3x25mm screws with M3 nuts	8	0.14*	-
M4x16mm screws with M4 nuts	8	0.17*	-

*Estimated prices

2.3 **Build instructions**

The parts, designed in FreeCAD, are 3D printed in PLA material, together with 50mm diameter PVC pipe sections, and nuts and bolts for fastening. The robotized distributor arm, measuring 552x497x236mm, uses two stepper motors for positioning, one on its base and another closer to the end of the arm, allowing it to effectively sweep a truncated circular sector below the pelletizer that covers the entire area of a tray. A third stepper actuates the opening and closing of a butterfly valve. The whole assembly is suspended beneath the pelletizer by two arms on the top funnel/hopper, inserted into aluminum railings just under where the pellets exit. These make their way through the first joint, the valve (if open), and through the second joint, before reaching the nozzle and dropping into a cup or being discarded. The tray holding the cups is set up on a scale, which enables the control of fill levels of cups, as well as determining the material yield.

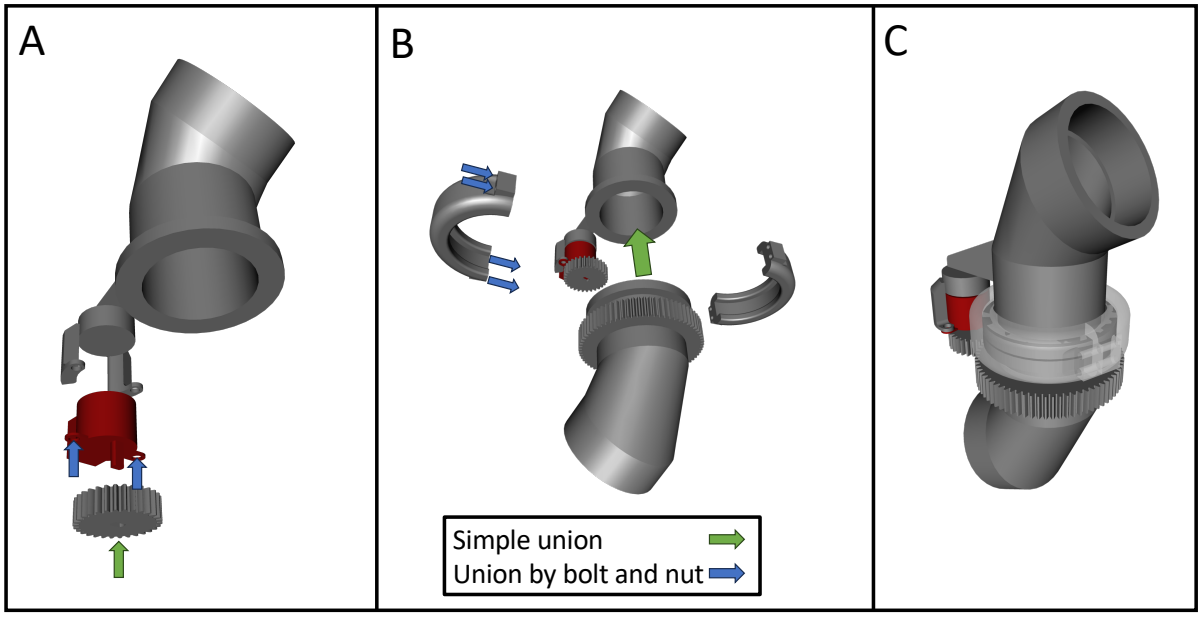

Figure S6: Assembly of the lower joint: A) Insert the motor shaft into the JointPinion, then fasten the motor to the linking part of Elbow2 using 2 M3x10mm screws. B) Align Elbow2 and Elbow3 so that they touch and the gears engage, then assemble two HalfClamp parts with 4 M3x25mm screws. C) Transparency view of the full lower joint assembly.

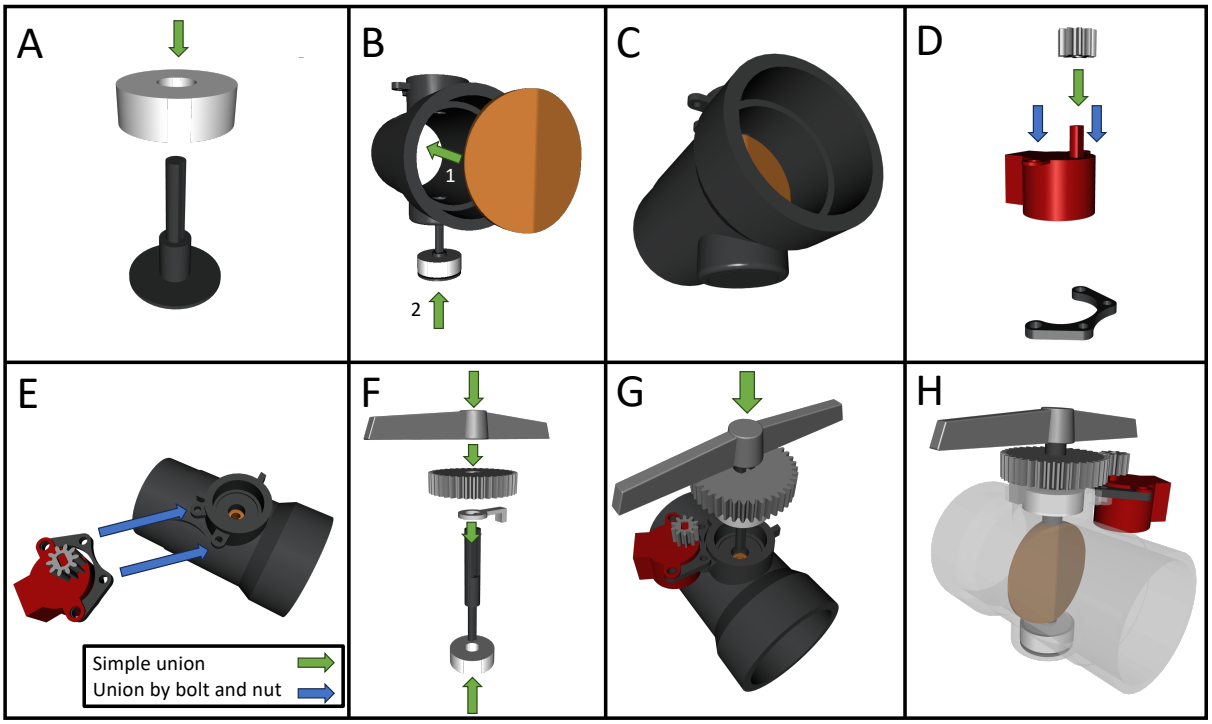

Figure S7: Assembly of the valve: A) Insert the Pin into the bearing. B) Place the Disk inside the ValveHousing, then pass the Pin through the Disc, pressure-fitting the bottom bearing into the housing. C) View of the current state of the assembly. D) Insert the ValvePinion into the motor shaft, then mount the MotorLink using 2 M3x10mm screws. E) Mount the motor assembly to the housing using 2 M3x10mm screws. F) Assemble the valve shaft by fitting the bearing into the Shaft, then the BallStop, the Gear and the Handle. G) Insert the shaft assembly into the housing and the disc, pressure fitting the bearing. H) Transparency view of the full valve assembly.

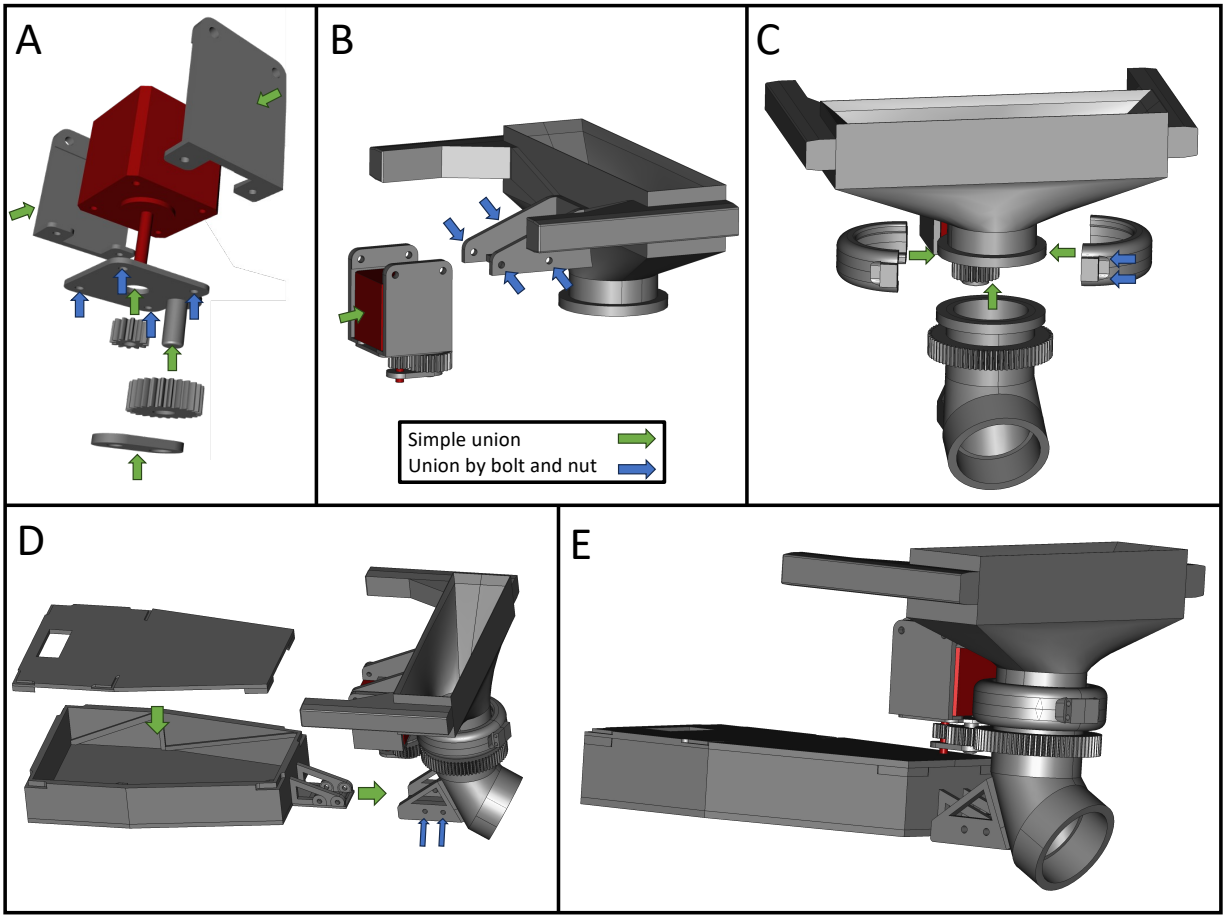

Figure S8: Assembly of the upper joint. A) Place the two NemaMount parts around the Nema17 motor. Then, secure the IdlerLink using 4 M3x10mm screws. After that, introduce the 1mmPinion in the motor shaft and the 1mmIdler in the IdlerLink shaft. Lastly, Introduce the IdlerCap. B) The motor assembly can then be joint to the TopFunnel using 4 M4x16mm screws. C) Use two HalfClamps to join the TopFunnel with Elbow1 using 4 M3x25mm screws. D) The CircuitCase and its Cover can be joined to Elbow1 using 4 M4x16mm screws. E) Full upper joint assembly.

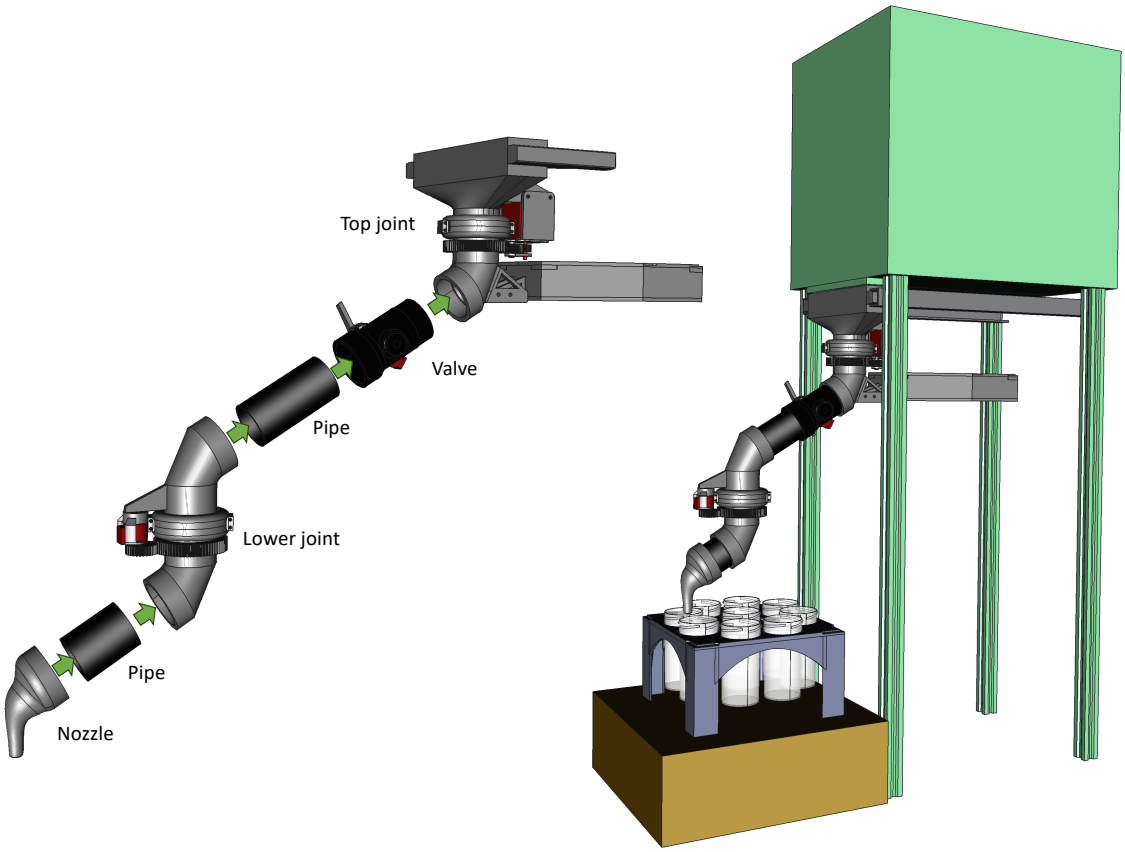

Figure S9: Final steps. On the left, all the previously assembled parts are combined. On the right, image of the full assembly mounted on the pelletizer (green) to fill the cups from the Tray (blue) that is set over the balance (brown).

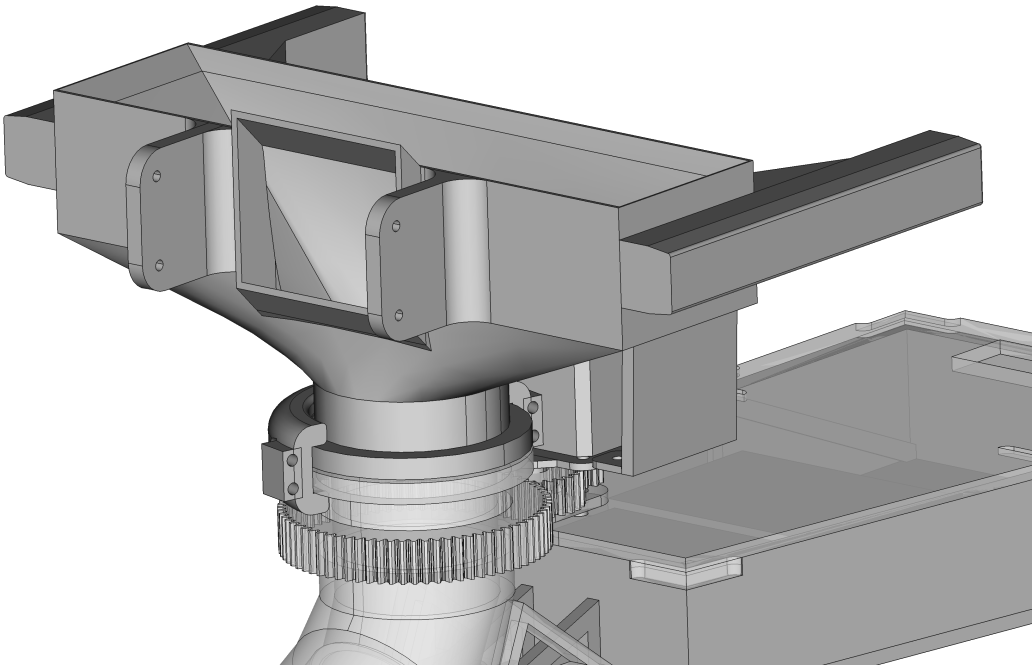

Figure S10: In the case that static charge is observed, the TopFunnel part can be exchanged for the TopFunnelIonizer version, that has a slot to add the ionizer included in the Bill of Materials.

2.4 **Circuit assembly**

The motors are controlled by A4988 drivers and an Arduino Uno R3 board, which is in turn managed by a Python script on a computer. The communication between these is established through a serial protocol, hence the Arduino must be plugged in by USB during operation, in addition to the power cable for the motors. The scale is also connected by its own serial port to the script. This Python code makes use of the Inverse Kinematics Python library (IKPy), through which the number of steps that each motor needs to take can be calculated in order for the robot to move its nozzle to a desired position. This versatility in the code, together with the fact that the arm should be able to reach any point within the swept region defined above, implies that the design is theoretically adaptable to any tray and cup arrangement, as long as these fit within said area.

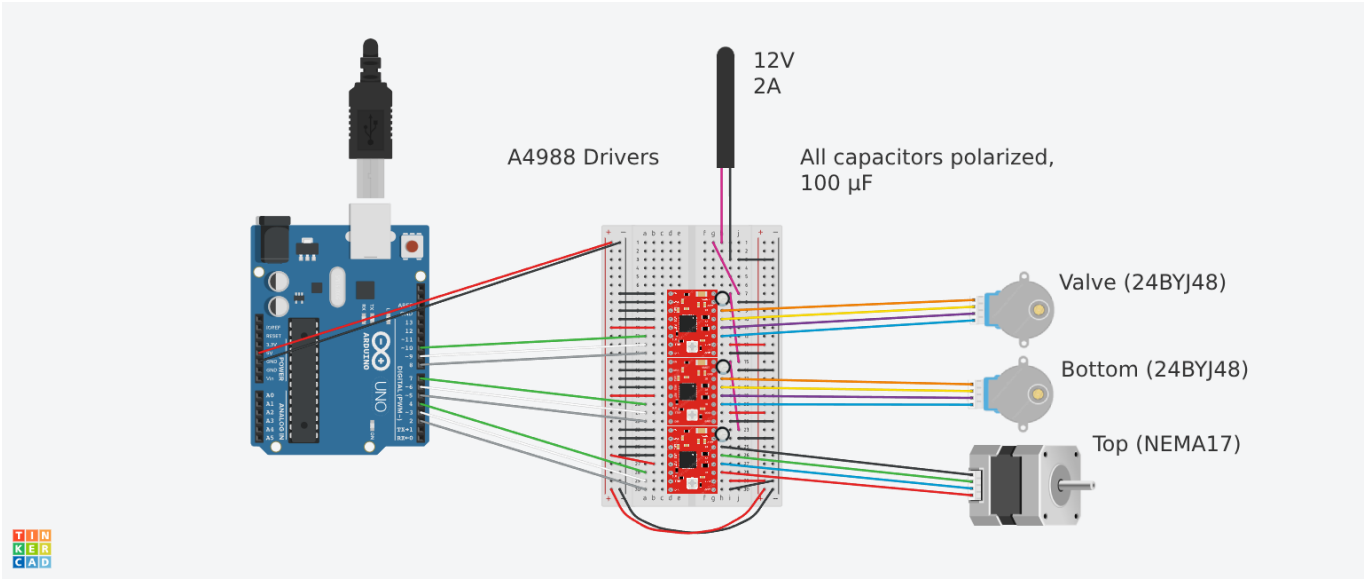

Figure S11: Schematics of the circuit. The external power source provides the $> 8V$ needed for the A4988 driver, in this case using a 12V wall power source. It's essential that it has common ground with the Arduino. Warning: don't power the circuit before ensuring that everything is properly connected, and don't modify connections if the circuit is powered.

2.5 Operation instructions

The device operates as follows: 1. Initial Setup: The robotized distributor arm must be positioned straight and perpendicular to the face of the pelletizer. The 9-hole square tray, aligned with the axis of rotation of the bottom joint, is placed on the scale. 2. Running the Python Script: Once the initial setup is complete, the Python script available in the project repository 31 is executed. The scale is automatically tared before the arm begins its operation. 3. Pellet Dispensing Process: The arm rotates to the coordinates of the first cup. The valve opens, allowing pellets to flow into the cup. When the predetermined weight capacity of the cup is reached, as measured by the scale, the valve closes. The arm then moves to the next cup. 4. Sequential Filling: This process continues, rotating through all 8 external cups on the tray sequentially, and finally filling the center cup. 5. Completion: After the center cup is filled, the arm returns to its initial position.

The following subsection reproduces the tutorial notebook available in the github repository.

2.5.1 Example using pellet distributor

This notebook contains an example of how to run the code for distributing the material. Most cells can be run without any modification if the distributor has been built without design modifications.

```
# Import Libraries
import serial
import numpy as np
import time
import os
import serial.tools.list_ports
```

In this cell, we define the port name for the Arduino and the balance. We do it by listing the connected ports and selecting the ones whose device serial is the corresponding to our device. You should find the serial of your devices and change it accordingly. You can do it by inspecting the devices detected by `serial.tools.list_ports.comports()`. In case of doubt, try disconnecting and connecting to see which ports are affected.

```
ports = serial.tools.list_ports.comports()

##########################################################################
```

```

arduino_serial = "write_here_your_arduino_serial"
balance_serial = "write_here_your_balance_serial"

######################################################################
for port in ports:
    if port.serial_number == arduino_serial:
        arduino_port = port.device
    elif port.serial_number == balance_serial:
        balance_port = port.device

### Serial initialization
## Arduino
arduino = serial.Serial(arduino_port, 9600)
## Balance
balance = serial.Serial(balance_port, 9600)
```

We import the functions from the pellet_distributor.py script. Note: you need to install the `ikpy` library

```
import pellet_distributor as ps
```

Define the capacity of the cup in kg

```
######################################################################
cup_capacity = 0.16
######################################################################
```

The next cell defines a series of parameters needed for the calculations. They don't have to be modified unless you have modified something in the design.

```
### Initializations

n_joints = 2
distributor_parameters = [{'steps_in_rotation':0,'gear_ratio':0,'length_mm':0,'bound_rad':0,
    'cartesian':[0,0]} for k in range(n_joints)]
## Top
distributor_parameters[0]['steps_in_rotation'] = np.round(200*1.1)   # Calibrate
    multiplicative factor to compensate lost steps if necessary
distributor_parameters[0]['gear_ratio'] = 6
distributor_parameters[0]['length_mm'] = 180.5
distributor_parameters[0]['bound_rad'] = np.deg2rad(35)    # np.deg2rad(35)  # IK Rotation
    bound in either direction, not total
distributor_parameters[0]['cartesian'] = [distributor_parameters[0].get('length_mm'), 0]
## Bot
distributor_parameters[1]['steps_in_rotation'] = 812
distributor_parameters[1]['gear_ratio'] = 2.4
distributor_parameters[1]['length_mm'] = np.sqrt(32.7**2 + 79.2**2)  # 84.4
distributor_parameters[1]['bound_rad'] = np.deg2rad(10*360)
distributor_parameters[1]['cartesian'] = [distributor_parameters[1].get('length_mm'), 0]




# Positions of the cups in the tray

tray_1 = ps.Tray('1', [distributor_parameters[0]['length_mm'], 0], [
    ps.Slot([79.2,32.7]),
    ps.Slot([32.7,79.2]),
    ps.Slot([-32.7,79.2]),
    ps.Slot([-79.2,32.7]),
    ps.Slot([-79.2,-32.7]),
    ps.Slot([-32.7,-79.2]),
    ps.Slot([32.7,-79.2]),
    ps.Slot([79.2,-32.7]),
    ps.Slot([0,0]),
```

```
    ])

tray_1.fill_all_slots_ez(0, cup_capacity)
tray_1.set_all_material('PLA####')
```

In the next cell, we use the Arduino Command Line Interface (arduino-cli) to compile and upload the arduino script. If you prefer to do it using the Graphical User Interface, skip this cell. For using it, update the location of your arduino-cli

```
######################################################################

os.system("/PATH/arduino-cli compile --fqbn arduino:avr:uno ../../Arduino/
    Pellet_distributor_v2/Pellet_distributor_v2.ino")
os.system("/PATH/arduino-cli upload -p "+arduino_port+" --fqbn arduino:avr:uno ../../Arduino/
    Pellet_distributor_v2/Pellet_distributor_v2.ino")

######################################################################
```

The `fill\_tray` function will run all the process. It uses the objects defined in the previous steps. The three parameters that allow interesting customization are:

`:param seq: List with the order of filling of the cups. If None, they will all be filled in default order`

`:param purge_slot: Number of the slot used for purging. This slot can be left without cup so the material is discarded. Used when changing material. If None, no purging will be done.`

`:param purge_time: Time for purge in minutes. Default: 0`

If `seq = None` and `purge_slot = None`, the cups will be filled sequentially (from 0 to 8) without any purging.

If `purge_slot` is not None, like `purge_slot = 0` for example, that hole will be used for purging. That hole should be left empty (without cup), and the tray should be in some kind of box or container so the pellets are not spilled everywhere. In that case, if seq is None the purging will be made after each cup, so the cup 1 will be filled, then purge, then cup 2, then purge, etc.

For more detailed control, you can use `seq`. For example, if you want cups 1 and 2 filled with one material and cups 3 and 4 with another, you can use `purge_slot = 0` and `seq = [1,2,0,3,4,0]`. In this way, after filling cups 1 and 2 with the same material it will make a purge in slot 0, and the fill cups 3 and 4.

```
######################################################################

ps.fill_tray(tray_1, distributor_parameters, arduino, balance, seq = None, purge_slot = 0,
    purge_time = 1)

######################################################################
```

NOTE: As an alternative, you can use `fill_tray_SIMULATED` to check if the filling order is the desired

```

ps.fill_tray_SIMULATED(tray_1, distributor_parameters, seq = None, purge_slot = 7, purge_time
    = 1)
```