import FreeCAD, Part, math

from FreeCAD import Vector

# Create a new document

doc = FreeCAD.newDocument("Kilimo360\_HydroponicSystem")

# System dimensions

width = 150

height = 200

depth = 40

num\_channels = 4

channel\_spacing = 40

channel\_width = 120

channel\_height = 8

channel\_depth = 25

pipe\_radius = 3

tank\_width = 50

tank\_height = 20

tank\_depth = 30

# --- Functions for creating objects ---

def create\_box(width, depth, height, vector):

"""Helper function to create a box."""

try:

box = Part.makeBox(width, depth, height, vector)

return box

except Exception as e:

FreeCAD.Console.PrintError(f"Error creating box: {e}\n")

return None

def create\_cylinder(radius, height, vector, direction=Vector(0, 0, 1)):

"""Helper function to create a cylinder."""

try:

cylinder = Part.makeCylinder(radius, height, vector, direction)

return cylinder

except Exception as e:

FreeCAD.Console.PrintError(f"Error creating cylinder: {e}\n")

return None

def create\_cone(radius1, radius2, height, vector, direction=Vector(0, 0, 1)):

"""Helper function to create a cone."""

try:

cone = Part.makeCone(radius1, radius2, height, vector, direction)

return cone

except Exception as e:

FreeCAD.Console.PrintError(f"Error creating cone: {e}\n")

return None

def create\_object(shape, name):

"""Helper function to create a FreeCAD object."""

try:

obj = doc.addObject("Part::Feature", name)

obj.Shape = shape

return obj

except Exception as e:

FreeCAD.Console.PrintError(f"Error creating object {name}: {e}\n")

return None

# --- Create hydroponic system components ---

# Nutrient solution tank at top

tank = create\_box(tank\_width, tank\_depth, tank\_height,

Vector(width/2 - tank\_width/2, depth/2 - tank\_depth/2, height - tank\_height))

tank\_obj = create\_object(tank, "NutrientTank")

# Nutrient mixer in tank

mixer\_height = 8

mixer\_radius = 5

mixer = create\_cylinder(mixer\_radius, mixer\_height,

Vector(width/2, depth/2, height - tank\_height + 5))

mixer\_blade1 = create\_box(mixer\_radius\*2, 1, 2,

Vector(width/2 - mixer\_radius, depth/2 - 0.5, height - tank\_height + mixer\_height/2))

mixer\_blade2 = create\_box(1, mixer\_radius\*2, 2,

Vector(width/2 - 0.5, depth/2 - mixer\_radius, height - tank\_height + mixer\_height/2))

mixer\_shape = None

if mixer and mixer\_blade1 and mixer\_blade2:

mixer\_shape = mixer.fuse(mixer\_blade1).fuse(mixer\_blade2)

mixer\_obj = create\_object(mixer\_shape, "NutrientMixer")

# pH sensor in tank

ph\_sensor\_width = 2

ph\_sensor\_depth = 2

ph\_sensor\_height = 12

ph\_sensor = create\_box(ph\_sensor\_width, ph\_sensor\_depth, ph\_sensor\_height,

Vector(width/2 - tank\_width/4, depth/2 - tank\_depth/4, height - tank\_height))

ph\_sensor\_obj = create\_object(ph\_sensor, "pHSensor")

# Nutrient dosing system

doser\_height = 15

doser\_radius = 3

doser = create\_cylinder(doser\_radius, doser\_height,

Vector(width/2 - tank\_width/4, depth/2, height - tank\_height - doser\_height))

doser\_nozzle = create\_cone(doser\_radius, doser\_radius/2, 5,

Vector(width/2 - tank\_width/4, depth/2, height - tank\_height))

doser\_shape = None

if doser and doser\_nozzle:

doser\_shape = doser.fuse(doser\_nozzle)

doser\_obj = create\_object(doser\_shape, "NutrientDoser")

# Main vertical pipe on right side (down)

main\_pipe\_down = create\_cylinder(pipe\_radius, height - tank\_height - 10,

Vector(width - 20, depth/2, 10))

# Main vertical pipe on left side (up)

main\_pipe\_up = create\_cylinder(pipe\_radius, height - tank\_height - 10,

Vector(20, depth/2, 10))

# Bottom connecting pipe

bottom\_pipe = create\_cylinder(pipe\_radius, width - 40,

Vector(20, depth/2, 10),

Vector(1, 0, 0))

# Pump (positioned at bottom)

pump\_length = 25

pump\_width = 15

pump\_height = 12

pump = create\_box(pump\_length, pump\_width, pump\_height,

Vector(width/2 - pump\_length/2, depth/2 - pump\_width/2, 5))

pump\_obj = create\_object(pump, "Pump")

# Water filter near pump

filter\_radius = 5

filter\_height = 15

filter = create\_cylinder(filter\_radius, filter\_height,

Vector(width/2 - pump\_length, depth/2, 10))

filter\_obj = create\_object(filter, "WaterFilter")

# Connecting pipe from pump to bottom pipe

pump\_connector = create\_cylinder(pipe\_radius, 10,

Vector(width/2, depth/2, pump\_height + 5),

Vector(1, 0, 0))

# Create connecting pipe from tank to main vertical pipe

tank\_outlet\_x = width/2 + tank\_width/2 # Right side of tank

tank\_outlet\_y = depth/2

tank\_outlet\_z = height - tank\_height/2 # Middle of tank height

# Horizontal pipe from tank to vertical pipe

tank\_connector\_h = create\_cylinder(

pipe\_radius,

width - 20 - tank\_outlet\_x, # Distance from tank to vertical pipe

Vector(tank\_outlet\_x, tank\_outlet\_y, tank\_outlet\_z),

Vector(1, 0, 0)

)

# Add a simple valve representation

valve\_radius = pipe\_radius \* 1.5

valve\_length = 8

valve\_position = Vector(tank\_outlet\_x + 10, tank\_outlet\_y, tank\_outlet\_z)

valve = create\_cylinder(valve\_radius, valve\_length, valve\_position, Vector(1, 0, 0))

valve\_handle = create\_box(2, 10, 2, Vector(valve\_position.x + valve\_length/2 - 1, valve\_position.y - 5, valve\_position.z + valve\_radius))

valve\_shape = None

if valve and valve\_handle:

valve\_shape = valve.fuse(valve\_handle)

valve\_obj = create\_object(valve\_shape, "FlowControlValve")

# Support structure (frame)

frame\_thickness = 3

# Vertical supports

left\_support = create\_box(frame\_thickness, frame\_thickness, height,

Vector(10, 10, 0))

right\_support = create\_box(frame\_thickness, frame\_thickness, height,

Vector(width - 10 - frame\_thickness, 10, 0))

# Add back vertical supports for more stability

back\_left\_support = create\_box(frame\_thickness, frame\_thickness, height,

Vector(10, depth - 10 - frame\_thickness, 0))

back\_right\_support = create\_box(frame\_thickness, frame\_thickness, height,

Vector(width - 10 - frame\_thickness, depth - 10 - frame\_thickness, 0))

frame\_supports = None

if left\_support and right\_support and back\_left\_support and back\_right\_support:

frame\_supports = left\_support.fuse(right\_support).fuse(back\_left\_support).fuse(back\_right\_support)

for i in range(num\_channels + 1): # +1 for the top support

if i == num\_channels:

# Top support for tank

channel\_y\_pos = height - tank\_height/2

else:

channel\_y\_pos = height - tank\_height - 20 - (i \* channel\_spacing)

# Front horizontal support

horizontal\_support\_front = create\_box(width - 20, frame\_thickness, frame\_thickness,

Vector(10, 10, channel\_y\_pos - frame\_thickness))

# Back horizontal support for additional stability

horizontal\_support\_back = create\_box(width - 20, frame\_thickness, frame\_thickness,

Vector(10, depth - 10 - frame\_thickness, channel\_y\_pos - frame\_thickness))

# Side horizontal supports to complete the frame

horizontal\_support\_left = create\_box(frame\_thickness, depth - 20, frame\_thickness,

Vector(10, 10, channel\_y\_pos - frame\_thickness))

horizontal\_support\_right = create\_box(frame\_thickness, depth - 20, frame\_thickness,

Vector(width - 10 - frame\_thickness, 10, channel\_y\_pos - frame\_thickness))

if horizontal\_support\_front and horizontal\_support\_back and horizontal\_support\_left and horizontal\_support\_right and frame\_supports:

frame\_supports = frame\_supports.fuse(horizontal\_support\_front).fuse(horizontal\_support\_back)

frame\_supports = frame\_supports.fuse(horizontal\_support\_left).fuse(horizontal\_support\_right)

if frame\_supports:

frame\_obj = create\_object(frame\_supports, "SupportFrame")

# Create growing channels

channels = []

channel\_connectors\_right = []

channel\_connectors\_left = []

plant\_holders = []

soil\_moisture\_sensors = []

drainage\_pipes = []

for i in range(num\_channels):

channel\_y\_pos = height - tank\_height - 20 - (i \* channel\_spacing)

# Create channel as a box

channel = create\_box(channel\_width, channel\_depth, channel\_height,

Vector((width - channel\_width)/2, (depth - channel\_depth)/2, channel\_y\_pos))

if channel:

channels.append(channel)

channel\_obj = create\_object(channel, f"Channel\_{i+1}")

# Create connecting pipes to main vertical pipes

if i % 2 == 0: # Even channels connect to right pipe

connector = create\_cylinder(pipe\_radius, (width - channel\_width)/2,

Vector(width - 20, depth/2, channel\_y\_pos + channel\_height/2),

Vector(-1, 0, 0))

if connector:

channel\_connectors\_right.append(connector)

else: # Odd channels connect to left pipe

connector = create\_cylinder(pipe\_radius, (width - channel\_width)/2,

Vector(20, depth/2, channel\_y\_pos + channel\_height/2),

Vector(1, 0, 0))

if connector:

channel\_connectors\_left.append(connector)

# Small drainage pipe at the end of each channel

if i % 2 == 0: # Even channels drain to left

drainage\_pipe = create\_cylinder(pipe\_radius/2, 10,

Vector((width - channel\_width)/2,

depth/2,

channel\_y\_pos),

Vector(0, 0, -1))

else: # Odd channels drain to right

drainage\_pipe = create\_cylinder(pipe\_radius/2, 10,

Vector((width + channel\_width)/2,

depth/2,

channel\_y\_pos),

Vector(0, 0, -1))

if drainage\_pipe:

drainage\_pipes.append(drainage\_pipe)

# Create plant holders (3 per channel)

for j in range(3):

plant\_x\_pos = (width - channel\_width)/2 + 20 + j \* (channel\_width - 40)/2

plant\_holder = create\_cylinder(5, 5,

Vector(plant\_x\_pos, depth/2, channel\_y\_pos + channel\_height))

# Create simple plant representation (stem and leaves)

stem\_height = 15

stem = create\_cylinder(1, stem\_height,

Vector(plant\_x\_pos, depth/2, channel\_y\_pos + channel\_height + 5))

# Create leaves using cones

leaf1 = create\_cone(0, 4, 8,

Vector(plant\_x\_pos, depth/2, channel\_y\_pos + channel\_height + 10),

Vector(1, 0, 1))

leaf2 = create\_cone(0, 4, 8,

Vector(plant\_x\_pos, depth/2, channel\_y\_pos + channel\_height + 10),

Vector(-1, 0, 1))

plant\_shape = None

if leaf1 and leaf2 and stem and plant\_holder:

plant\_shape = plant\_holder.fuse(stem)

if leaf1:

plant\_shape = plant\_shape.fuse(leaf1)

if leaf2:

plant\_shape = plant\_shape.fuse(leaf2)

if plant\_shape:

plant\_holders.append(plant\_shape)

holder\_obj = create\_object(plant\_shape, f"PlantHolder\_{i\*3+j+1}")

# Add soil moisture sensor next to first plant in each channel

if j == 0:

sensor\_width = 2

sensor\_height = 10

sensor\_depth = 1

moisture\_sensor = create\_box(sensor\_width, sensor\_depth, sensor\_height,

Vector(plant\_x\_pos + 8, depth/2, channel\_y\_pos))

if moisture\_sensor:

soil\_moisture\_sensors.append(moisture\_sensor)

sensor\_obj = create\_object(moisture\_sensor, f"SoilMoistureSensor\_{i+1}")

# Add all pipes to document

pipe\_system = None

if main\_pipe\_down and main\_pipe\_up:

pipe\_system = main\_pipe\_down.fuse(main\_pipe\_up)

if bottom\_pipe:

pipe\_system = pipe\_system.fuse(bottom\_pipe)

if pump\_connector:

pipe\_system = pipe\_system.fuse(pump\_connector)

# Add the tank connector and valve to the pipe system

if tank\_connector\_h:

pipe\_system = pipe\_system.fuse(tank\_connector\_h)

# Add drainage pipes to the system

for drainage\_pipe in drainage\_pipes:

if drainage\_pipe and pipe\_system:

pipe\_system = pipe\_system.fuse(drainage\_pipe)

# Add channel connectors

for connector in channel\_connectors\_right + channel\_connectors\_left:

if connector and pipe\_system:

pipe\_system = pipe\_system.fuse(connector)

if pipe\_system:

pipe\_obj = create\_object(pipe\_system, "PipeSystem")

# Modified IoT components enclosure - smaller size

iot\_enclosure\_width = 45 # Reduced from 80

iot\_enclosure\_depth = 25 # Reduced from 40

iot\_enclosure\_height = 25 # Reduced from 30

iot\_enclosure\_wall = 2

# Main IoT enclosure (at the side of the system)

iot\_outer = create\_box(iot\_enclosure\_width, iot\_enclosure\_depth, iot\_enclosure\_height,

Vector(width + 10, depth/2 - iot\_enclosure\_depth/2, height/2))

iot\_inner = create\_box(iot\_enclosure\_width - 2\*iot\_enclosure\_wall,

iot\_enclosure\_depth - 2\*iot\_enclosure\_wall,

iot\_enclosure\_height - iot\_enclosure\_wall,

Vector(width + 10 + iot\_enclosure\_wall,

depth/2 - iot\_enclosure\_depth/2 + iot\_enclosure\_wall,

height/2 + iot\_enclosure\_wall))

iot\_enclosure = None

if iot\_outer and iot\_inner:

iot\_enclosure = iot\_outer.cut(iot\_inner)

iot\_enclosure\_obj = create\_object(iot\_enclosure, "IoTEnclosure")

# Lid for IoT enclosure

iot\_lid = create\_box(iot\_enclosure\_width, iot\_enclosure\_depth, iot\_enclosure\_wall,

Vector(width + 10, depth/2 - iot\_enclosure\_depth/2, height/2 + iot\_enclosure\_height))

iot\_lid\_obj = None

if iot\_lid:

iot\_lid\_obj = create\_object(iot\_lid, "IoTLid")

# FIX: Proper hinge for IoT enclosure that attaches to both enclosure and lid

iot\_hinge\_radius = iot\_enclosure\_wall/2

iot\_hinge\_length = iot\_enclosure\_depth

# Create the hinge cylinder

iot\_hinge = create\_cylinder(iot\_hinge\_radius, iot\_hinge\_length,

Vector(width + 10, depth/2 + iot\_enclosure\_depth/2 - iot\_hinge\_radius,

height/2 + iot\_enclosure\_height),

Vector(0, -1, 0))

# Create attachment pieces to connect the hinge to the lid and enclosure

hinge\_attach\_width = iot\_hinge\_radius \* 3

hinge\_attach\_height = iot\_hinge\_radius \* 3

hinge\_attach\_depth = iot\_hinge\_radius \* 2

# Top attachment (to lid)

top\_attach = create\_box(hinge\_attach\_width, iot\_hinge\_length, hinge\_attach\_height,

Vector(width + 10 - hinge\_attach\_width/2,

depth/2 - iot\_enclosure\_depth/2,

height/2 + iot\_enclosure\_height - hinge\_attach\_height/2))

# Side attachment (to enclosure)

side\_attach = create\_box(hinge\_attach\_width, hinge\_attach\_depth, hinge\_attach\_height \* 2,

Vector(width + 10 - hinge\_attach\_width/2,

depth/2 - iot\_enclosure\_depth/2,

height/2 + iot\_enclosure\_height - hinge\_attach\_height \* 1.5))

# Combine hinge components

iot\_hinge\_assembly = None

if iot\_hinge and top\_attach and side\_attach:

iot\_hinge\_assembly = iot\_hinge.fuse(top\_attach).fuse(side\_attach)

iot\_hinge\_obj = create\_object(iot\_hinge\_assembly, "IoTHinge")

# ESP32 inside enclosure

esp32\_width = 30

esp32\_depth = 20

esp32\_height = 5

esp32 = create\_box(esp32\_width, esp32\_depth, esp32\_height,

Vector(width + 15, depth/2 - esp32\_depth/2, height/2 + 5))

if esp32:

esp32\_obj = create\_object(esp32, "ESP32")

# DHT11 inside enclosure - moved closer to ESP32

dht11\_width = 15

dht11\_depth = 10

dht11\_height = 5

dht11 = create\_box(dht11\_width, dht11\_depth, dht11\_height,

Vector(width + 15, depth/2 - dht11\_depth/2, height/2 + 12))

if dht11:

dht11\_obj = create\_object(dht11, "DHT11")

# Relay module inside enclosure - stacked on top

relay\_width = 25

relay\_depth = 15

relay\_height = 5

relay = create\_box(relay\_width, relay\_depth, relay\_height,

Vector(width + 15, depth/2 - relay\_depth/2, height/2 + 19))

if relay:

relay\_obj = create\_object(relay, "RelayModule")

# LCD Display inside enclosure - placed on the side

lcd\_width = 35

lcd\_depth = 3

lcd\_height = 15

lcd = create\_box(lcd\_width, lcd\_depth, lcd\_height,

Vector(width + 10 + iot\_enclosure\_wall/2, depth/2 - lcd\_height/2, height/2 + 5))

if lcd:

lcd\_obj = create\_object(lcd, "LCDDisplay")

# LED alert - repositioned to the new enclosure size

led\_radius = 2

led\_height = 5

led = create\_cylinder(led\_radius, led\_height,

Vector(width + 10 + iot\_enclosure\_width - 10, depth/2 + 10, height/2 + iot\_enclosure\_height - led\_height))

if led:

led\_obj = create\_object(led, "LEDAlert")

# Water level sensor in tank

water\_level\_width = 2

water\_level\_depth = 2

water\_level\_height = 15

water\_level\_sensor = create\_box(water\_level\_width, water\_level\_depth, water\_level\_height,

Vector(width/2, depth/2, height - tank\_height))

if water\_level\_sensor:

water\_level\_obj = create\_object(water\_level\_sensor, "WaterLevelSensor")

# Conduit for wiring - updated to match the new enclosure position

conduit\_radius = 3

conduit = create\_cylinder(conduit\_radius, 20,

Vector(width, depth/2, height/2 + iot\_enclosure\_height/2),

Vector(-1, 0, 0))

if conduit:

conduit\_obj = create\_object(conduit, "WiringConduit")

# Add waterproof cover/seal for the wiring conduit

conduit\_seal\_radius = conduit\_radius \* 1.5

conduit\_seal\_height = 2

conduit\_seal = create\_cylinder(conduit\_seal\_radius, conduit\_seal\_height,

Vector(width - 2, depth/2, height/2 + iot\_enclosure\_height/2),

Vector(-1, 0, 0))

if conduit\_seal:

conduit\_seal\_obj = create\_object(conduit\_seal, "ConduitSeal")

# Solar panel on top with improved mount

solar\_width = 40

solar\_height = 2

solar\_depth = 30

# Create a mount for the solar panel

solar\_mount\_height = 15

solar\_mount\_width = 5

solar\_mount\_depth = 5

solar\_mount = create\_box(solar\_mount\_width, solar\_mount\_depth, solar\_mount\_height,

Vector(width + 10 + solar\_width/2 - solar\_mount\_width/2,

depth/2,

height + 10 - solar\_mount\_height))

# Create solar panel

solar\_panel = create\_box(solar\_width, solar\_depth, solar\_height,

Vector(width + 10, depth/2 - solar\_depth/2, height + 10))

# Combine mount and panel

solar\_assembly = None

if solar\_panel and solar\_mount:

solar\_assembly = solar\_panel.fuse(solar\_mount)

solar\_obj = create\_object(solar\_assembly, "SolarPanel")

# Add backup battery in IoT enclosure

battery\_width = 20

battery\_height = 8

battery\_depth = 10

battery = create\_box(battery\_width, battery\_depth, battery\_height,

Vector(width + 20, depth/2 + 5, height/2 + 5))

if battery:

battery\_obj = create\_object(battery, "BackupBattery")

# Set colors for better visualization

# We need to import FreeCADGui for this

try:

import FreeCADGui as Gui

# Water/liquid objects in blue

if 'tank\_obj' in locals() and tank\_obj:

tank\_obj.ViewObject.ShapeColor = (0.0, 0.5, 1.0)

if 'pipe\_obj' in locals() and pipe\_obj:

pipe\_obj.ViewObject.ShapeColor = (0.7, 0.7, 0.7)

# Electronics in dark colors

if 'esp32\_obj' in locals() and esp32\_obj:

esp32\_obj.ViewObject.ShapeColor = (0.2, 0.2, 0.2)

if 'relay\_obj' in locals() and relay\_obj:

relay\_obj.ViewObject.ShapeColor = (0.3, 0.3, 0.3)

if 'lcd\_obj' in locals() and lcd\_obj:

lcd\_obj.ViewObject.ShapeColor = (0.1, 0.1, 0.3)

if 'battery\_obj' in locals() and battery\_obj:

battery\_obj.ViewObject.ShapeColor = (0.5, 0.0, 0.0)

# Plants in green

for i in range(num\_channels \* 3):

obj = doc.getObject(f"PlantHolder\_{i+1}")

if obj:

obj.ViewObject.ShapeColor = (0.0, 0.8, 0.0)

# Frame in dark grey

if 'frame\_obj' in locals() and frame\_obj:

frame\_obj.ViewObject.ShapeColor = (0.3, 0.3, 0.3)

# Solar panel in blue-black

if 'solar\_obj' in locals() and solar\_obj:

solar\_obj.ViewObject.ShapeColor = (0.1, 0.1, 0.2)

if 'iot\_enclosure\_obj' in locals() and iot\_enclosure\_obj:

iot\_enclosure\_obj.ViewObject.ShapeColor = (0.6, 0.3, 0.0)

if 'iot\_hinge\_obj' in locals() and iot\_hinge\_obj:

iot\_hinge\_obj.ViewObject.ShapeColor = (0.4, 0.4, 0.4)

except Exception as e:

FreeCAD.Console.PrintWarning(f"Error setting colors: {e}\n")

# Strengthen Pipe Supports

if 'pipe\_obj' in locals() and pipe\_obj:

pipe\_system = pipe\_obj.Shape

bracket\_thickness = 5

bracket\_height = 20

bracket\_width = 10

# Add brackets for both vertical pipes

for i in range(3): # Add three brackets for right pipe

bracket = create\_box(bracket\_width, bracket\_thickness, bracket\_height,

Vector(width - 30, depth/2 - bracket\_thickness/2, 10 + i \* (height - tank\_height)/3))

if bracket and pipe\_system:

pipe\_system = pipe\_system.fuse(bracket)

for i in range(3): # Add three brackets for left pipe

bracket = create\_box(bracket\_width, bracket\_thickness, bracket\_height,

Vector(20, depth/2 - bracket\_thickness/2, 10 + i \* (height - tank\_height)/3))

if bracket and pipe\_system:

pipe\_system = pipe\_system.fuse(bracket)

if pipe\_system:

pipe\_obj.Shape = pipe\_system

# Tank Connection Reinforcement

if 'pipe\_obj' in locals() and pipe\_obj and 'tank\_outlet\_x' in locals():

pipe\_system = pipe\_obj.Shape

flange\_radius = pipe\_radius \* 3

flange\_thickness = 2

flange = create\_cylinder(flange\_radius, flange\_thickness,

Vector(tank\_outlet\_x, tank\_outlet\_y, tank\_outlet\_z - flange\_thickness))

if flange and pipe\_system:

pipe\_system = pipe\_system.fuse(flange)

pipe\_obj.Shape = pipe\_system

# Rounded Edges for IoT enclosure

if 'iot\_enclosure\_obj' in locals() and iot\_enclosure\_obj:

try:

# Instead of trying to fillet all edges at once, select just a few key edges

edges\_to\_fillet = []

for i, edge in enumerate(iot\_enclosure\_obj.Shape.Edges):

if i < 12: # Just fillet some of the edges to avoid errors

edges\_to\_fillet.append(edge)

if edges\_to\_fillet:

iot\_enclosure\_fillet = iot\_enclosure\_obj.Shape.makeFillet(2, edges\_to\_fillet) # Fillet radius is 2mm

iot\_enclosure\_obj.Shape = iot\_enclosure\_fillet

except Exception as e:

FreeCAD.Console.PrintWarning(f"Fillet error: {e}\n")

# Ventilation Slots

if 'iot\_enclosure\_obj' in locals() and iot\_enclosure\_obj:

try:

iot\_shape = iot\_enclosure\_obj.Shape

slot\_width = 5

slot\_height = 1

slot\_spacing = 10

for i in range(3): # Reduce to 3 slots to avoid potential issues

slot = create\_box(slot\_width, slot\_height, iot\_enclosure\_wall,

Vector(width + 15 + i \* slot\_spacing,

depth/2 - iot\_enclosure\_depth/2 + slot\_spacing,

height/2 + iot\_enclosure\_wall))

if slot and iot\_shape:

iot\_shape = iot\_shape.cut(slot)

if iot\_shape:

iot\_enclosure\_obj.Shape = iot\_shape

except Exception as e:

FreeCAD.Console.PrintWarning(f"Ventilation slot error: {e}\n")

# Add channel drain plugs for maintenance

for i in range(num\_channels):

if i % 2 == 0: # Even channels drain to left

drain\_plug\_pos = Vector((width - channel\_width)/2, depth/2,

height - tank\_height - 20 - (i \* channel\_spacing))

else: # Odd channels drain to right

drain\_plug\_pos = Vector((width + channel\_width)/2, depth/2,

height - tank\_height - 20 - (i \* channel\_spacing))

drain\_plug = create\_cylinder(pipe\_radius, 3, drain\_plug\_pos, Vector(0, 0, -1))

drain\_handle = create\_cylinder(pipe\_radius\*1.5, 1,

Vector(drain\_plug\_pos.x, drain\_plug\_pos.y, drain\_plug\_pos.z - 3))

if drain\_plug and drain\_handle:

draindrain\_plug\_assembly = drain\_plug.fuse(drain\_handle)

drain\_plug\_obj = create\_object(drain\_plug\_assembly, f"DrainPlug\_{i+1}")

if 'drain\_plug\_obj' in locals() and drain\_plug\_obj:

try:

drain\_plug\_obj.ViewObject.ShapeColor = (0.8, 0.0, 0.0) # Red color for drain plugs

except:

pass

# Add water distribution system

for i in range(num\_channels):

channel\_y\_pos = height - tank\_height - 20 - (i \* channel\_spacing)

# Create distribution pipe along the channel

distrib\_pipe = create\_cylinder(pipe\_radius/1.5, channel\_width - 10,

Vector((width - channel\_width)/2 + 5, depth/2, channel\_y\_pos + channel\_height + 5),

Vector(1, 0, 0))

# Create water drippers

dripper\_radius = 2

dripper\_height = 2

dripper\_assembly = None

for j in range(3):

dripper\_x\_pos = (width - channel\_width)/2 + 20 + j \* (channel\_width - 40)/2

dripper = create\_cylinder(dripper\_radius, dripper\_height,

Vector(dripper\_x\_pos, depth/2, channel\_y\_pos + channel\_height + 5),

Vector(0, 0, -1))

if dripper and (j == 0 or dripper\_assembly):

if j == 0:

dripper\_assembly = dripper

else:

dripper\_assembly = dripper\_assembly.fuse(dripper)

# Connect distribution pipe to drippers

if distrib\_pipe and dripper\_assembly:

distrib\_system = distrib\_pipe.fuse(dripper\_assembly)

distrib\_obj = create\_object(distrib\_system, f"DistributionSystem\_{i+1}")

if 'distrib\_obj' in locals() and distrib\_obj:

try:

distrib\_obj.ViewObject.ShapeColor = (0.0, 0.5, 0.8) # Light blue for water distribution

except:

pass

# Add supporting legs

leg\_height = 50

leg\_width = 10

leg\_thickness = 10

# Create four legs

legs = []

leg\_positions = [

Vector(20, 20, -leg\_height),

Vector(width - 20 - leg\_width, 20, -leg\_height),

Vector(20, depth - 20 - leg\_thickness, -leg\_height),

Vector(width - 20 - leg\_width, depth - 20 - leg\_thickness, -leg\_height)

]

for i, pos in enumerate(leg\_positions):

leg = create\_box(leg\_width, leg\_thickness, leg\_height, pos)

if leg:

legs.append(leg)

leg\_obj = create\_object(leg, f"SupportLeg\_{i+1}")

if 'leg\_obj' in locals() and leg\_obj:

try:

leg\_obj.ViewObject.ShapeColor = (0.5, 0.35, 0.05) # Brown for wooden legs

except:

pass

# Add feet/base for stability

base\_height = 5

base\_width = 20

base\_depth = 20

for i, pos in enumerate(leg\_positions):

base = create\_box(base\_width, base\_depth, base\_height,

Vector(pos.x - (base\_width - leg\_width)/2,

pos.y - (base\_depth - leg\_thickness)/2,

pos.z))

if base:

base\_obj = create\_object(base, f"LegBase\_{i+1}")

if 'base\_obj' in locals() and base\_obj:

try:

base\_obj.ViewObject.ShapeColor = (0.3, 0.3, 0.3) # Dark grey for bases

except:

pass

# Add information label

label\_width = 30

label\_height = 15

label\_thickness = 1

label = create\_box(label\_width, label\_thickness, label\_height,

Vector(width/2 - label\_width/2, 5, height/2))

if label:

label\_obj = create\_object(label, "InfoLabel")

if 'label\_obj' in locals() and label\_obj:

try:

label\_obj.ViewObject.ShapeColor = (1.0, 1.0, 1.0) # White for label

except:

pass

# Add overflow protection system

overflow\_pipe\_radius = pipe\_radius

overflow\_height = 10

overflow\_width = 15

# Create overflow outlet from tank

overflow\_outlet = create\_cylinder(overflow\_pipe\_radius, overflow\_width,

Vector(width/2 - tank\_width/2,

depth/2,

height - tank\_height + 5),

Vector(-1, 0, 0))

if overflow\_outlet:

overflow\_obj = create\_object(overflow\_outlet, "OverflowProtection")

if 'overflow\_obj' in locals() and overflow\_obj:

try:

overflow\_obj.ViewObject.ShapeColor = (0.7, 0.7, 0.7) # Grey for overflow pipe

except:

pass

# Add access panel for maintenance

panel\_width = 30

panel\_height = 25

panel\_thickness = 2

# Create access panel on the side

access\_panel = create\_box(panel\_thickness, panel\_width, panel\_height,

Vector(width - 5, depth/2 - panel\_width/2, height/3))

if access\_panel:

panel\_obj = create\_object(access\_panel, "AccessPanel")

if 'panel\_obj' in locals() and panel\_obj:

try:

panel\_obj.ViewObject.ShapeColor = (0.8, 0.8, 0.8) # Light grey for panel

except:

pass

# Add hinges for access panel

hinge\_radius = 2

hinge\_length = 10

for i in range(2):

hinge\_pos\_z = height/3 + i \* panel\_height \* 0.8

hinge = create\_cylinder(hinge\_radius, hinge\_length,

Vector(width - 5, depth/2 - panel\_width/2, hinge\_pos\_z),

Vector(0, 1, 0))

if hinge:

hinge\_obj = create\_object(hinge, f"PanelHinge\_{i+1}")

if 'hinge\_obj' in locals() and hinge\_obj:

try:

hinge\_obj.ViewObject.ShapeColor = (0.6, 0.6, 0.6) # Medium grey for hinges

except:

pass

# Add Kilimo360 branding

try:

import Part

from FreeCAD import Base

# Create text as a Part shape

text = Part.makeWire(Part.makePolygon([

Base.Vector(width/2 - 40, 10, height - 20),

Base.Vector(width/2 + 40, 10, height - 20),

Base.Vector(width/2 + 40, 10, height - 10),

Base.Vector(width/2 - 40, 10, height - 10),

Base.Vector(width/2 - 40, 10, height - 20)

]))

if text:

text\_obj = doc.addObject("Part::Feature", "Kilimo360\_Brand")

text\_obj.Shape = text

try:

text\_obj.ViewObject.LineColor = (0.0, 0.7, 0.0) # Green for branding

text\_obj.ViewObject.LineWidth = 5

except:

pass

except:

pass

# Save the document

doc.recompute()

print("Hydroponic system model created successfully!")

# Display the model

try:

import FreeCADGui

FreeCADGui.ActiveDocument = FreeCADGui.getDocument(doc.Name)

FreeCADGui.SendMsgToActiveView("ViewFit")

FreeCADGui.activeDocument().activeView().viewIsometric()

except:

pass