import FreeCAD

import Part

from FreeCAD import Vector

# Create a new document

doc = FreeCAD.newDocument("IoT\_Enclosure")

# Helper 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\_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

# IoT enclosure dimensions

iot\_enclosure\_width = 45

iot\_enclosure\_depth = 25

iot\_enclosure\_height = 25

iot\_enclosure\_wall = 2

# Position (in new standalone model, we can center it)

position\_x = 0

position\_y = 0

position\_z = 0

# Main IoT enclosure

iot\_outer = create\_box(

iot\_enclosure\_width,

iot\_enclosure\_depth,

iot\_enclosure\_height,

Vector(position\_x, position\_y, position\_z)

)

# Inner cutout for the hollow part

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(

position\_x + iot\_enclosure\_wall,

position\_y + iot\_enclosure\_wall,

position\_z + iot\_enclosure\_wall

)

)

# Create the enclosure by cutting the inner box from the outer box

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(

position\_x,

position\_y,

position\_z + iot\_enclosure\_height

)

)

if iot\_lid:

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

# Hinge for IoT enclosure

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(

position\_x,

position\_y + iot\_enclosure\_depth - iot\_hinge\_radius,

position\_z + 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(

position\_x - hinge\_attach\_width/2,

position\_y,

position\_z + 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(

position\_x - hinge\_attach\_width/2,

position\_y,

position\_z + 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")

# Add internal components

# ESP32

esp32\_width = 30

esp32\_depth = 20

esp32\_height = 5

esp32 = create\_box(

esp32\_width,

esp32\_depth,

esp32\_height,

Vector(

position\_x + 5,

position\_y + (iot\_enclosure\_depth - esp32\_depth)/2,

position\_z + 5

)

)

if esp32:

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

# DHT11 temperature/humidity sensor

dht11\_width = 15

dht11\_depth = 10

dht11\_height = 5

dht11 = create\_box(

dht11\_width,

dht11\_depth,

dht11\_height,

Vector(

position\_x + 5,

position\_y + (iot\_enclosure\_depth - dht11\_depth)/2,

position\_z + 12

)

)

if dht11:

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

# Relay module

relay\_width = 25

relay\_depth = 15

relay\_height = 5

relay = create\_box(

relay\_width,

relay\_depth,

relay\_height,

Vector(

position\_x + 5,

position\_y + (iot\_enclosure\_depth - relay\_depth)/2,

position\_z + 19

)

)

if relay:

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

# LCD Display

lcd\_width = 35

lcd\_depth = 3

lcd\_height = 15

lcd = create\_box(

lcd\_width,

lcd\_depth,

lcd\_height,

Vector(

position\_x + iot\_enclosure\_wall/2,

position\_y + (iot\_enclosure\_depth - lcd\_height)/2,

position\_z + 5

)

)

if lcd:

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

# LED alert

led\_radius = 2

led\_height = 5

led = create\_cylinder(

led\_radius,

led\_height,

Vector(

position\_x + iot\_enclosure\_width - 10,

position\_y + iot\_enclosure\_depth - 5,

position\_z + iot\_enclosure\_height - led\_height

)

)

if led:

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

# Backup battery

battery\_width = 20

battery\_height = 8

battery\_depth = 10

battery = create\_box(

battery\_width,

battery\_depth,

battery\_height,

Vector(

position\_x + 10,

position\_y + iot\_enclosure\_depth - 15,

position\_z + 5

)

)

if battery:

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

# Wiring conduit for external connections

conduit\_radius = 3

conduit = create\_cylinder(

conduit\_radius,

iot\_enclosure\_wall \* 2,

Vector(

position\_x,

position\_y + iot\_enclosure\_depth/2,

position\_z + 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(

position\_x - conduit\_seal\_height,

position\_y + iot\_enclosure\_depth/2,

position\_z + iot\_enclosure\_height/2

),

Vector(-1, 0, 0)

)

if conduit\_seal:

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

# Add ventilation slots for cooling

try:

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

iot\_shape = iot\_enclosure\_obj.Shape

slot\_width = 15

slot\_height = 2

slot\_spacing = 6

# Create multiple ventilation slots

for i in range(3):

slot = create\_box(

slot\_width,

slot\_height,

iot\_enclosure\_wall,

Vector(

position\_x + 5 + i \* slot\_spacing,

position\_y + slot\_spacing,

position\_z + iot\_enclosure\_wall

)

)

if slot and iot\_shape:

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

# Create slots on another side for better airflow

for i in range(3):

slot = create\_box(

slot\_width,

slot\_height,

iot\_enclosure\_wall,

Vector(

position\_x + 5 + i \* slot\_spacing,

position\_y + iot\_enclosure\_depth - slot\_spacing - slot\_height,

position\_z + 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")

# Try to add rounded edges for better appearance

try:

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

# Select a subset of edges to fillet to avoid computational issues

edges\_to\_fillet = []

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

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

edges\_to\_fillet.append(edge)

if edges\_to\_fillet:

iot\_enclosure\_fillet = iot\_enclosure\_obj.Shape.makeFillet(2, edges\_to\_fillet)

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

except Exception as e:

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

# Add mounting holes for attaching the enclosure to the main system

try:

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

iot\_shape = iot\_enclosure\_obj.Shape

mount\_hole\_radius = 1.5

mount\_hole\_depth = iot\_enclosure\_wall \* 2

# Create four mounting holes, one at each corner

mount\_positions = [

Vector(position\_x + 7, position\_y + 7, position\_z),

Vector(position\_x + iot\_enclosure\_width - 7, position\_y + 7, position\_z),

Vector(position\_x + 7, position\_y + iot\_enclosure\_depth - 7, position\_z),

Vector(position\_x + iot\_enclosure\_width - 7, position\_y + iot\_enclosure\_depth - 7, position\_z)

]

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

mount\_hole = create\_cylinder(

mount\_hole\_radius,

mount\_hole\_depth,

pos,

Vector(0, 0, 1)

)

if mount\_hole and iot\_shape:

iot\_shape = iot\_shape.cut(mount\_hole)

if iot\_shape:

iot\_enclosure\_obj.Shape = iot\_shape

except Exception as e:

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

# Add snap-fit latches for securing the lid

try:

latch\_width = 5

latch\_height = 2

latch\_depth = 2

# Create latches on the front side

latch1 = create\_box(

latch\_width,

latch\_depth,

latch\_height,

Vector(

position\_x + iot\_enclosure\_width/2 - latch\_width/2,

position\_y + iot\_enclosure\_depth - latch\_depth,

position\_z + iot\_enclosure\_height - latch\_height/2

)

)

if latch1:

latch1\_obj = create\_object(latch1, "Latch1")

except Exception as e:

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

# Set colors for better visualization

try:

import FreeCADGui as Gui

# Main enclosure in orange

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

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

# Lid in slightly darker orange

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

iot\_lid\_obj.ViewObject.ShapeColor = (0.5, 0.25, 0.0)

# Hinge in metallic gray

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

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

# Electronics in appropriate colors

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

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

if 'dht11\_obj' in locals() and dht11\_obj:

dht11\_obj.ViewObject.ShapeColor = (0.0, 0.3, 0.6)

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 'led\_obj' in locals() and led\_obj:

led\_obj.ViewObject.ShapeColor = (1.0, 0.0, 0.0) # Red LED

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

battery\_obj.ViewObject.ShapeColor = (0.5, 0.0, 0.0) # Dark red for battery

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

conduit\_obj.ViewObject.ShapeColor = (0.7, 0.7, 0.7) # Light gray for conduit

if 'conduit\_seal\_obj' in locals() and conduit\_seal\_obj:

conduit\_seal\_obj.ViewObject.ShapeColor = (0.2, 0.2, 0.2) # Dark gray for seal

if 'latch1\_obj' in locals() and latch1\_obj:

latch1\_obj.ViewObject.ShapeColor = (0.5, 0.25, 0.0) # Match lid color

except Exception as e:

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

# Add a label for the enclosure

try:

label\_width = 30

label\_height = 5

label\_thickness = 0.2

label = create\_box(

label\_width,

label\_thickness,

label\_height,

Vector(

position\_x + (iot\_enclosure\_width - label\_width)/2,

position\_y - label\_thickness,

position\_z + iot\_enclosure\_height/2

)

)

if label:

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

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

except Exception as e:

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

# Save the document

doc.recompute()

print("IoT enclosure model created successfully!")

# Display the model

try:

import FreeCADGui

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

FreeCADGui.SendMsgToActiveView("ViewFit")

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

except:

pass

import FreeCAD

import Part

from FreeCAD import Vector

# Create a new document or use the existing one

try:

doc = FreeCAD.activeDocument()

if not doc:

doc = FreeCAD.newDocument("IoT\_Enclosure")

except:

doc = FreeCAD.newDocument("IoT\_Enclosure")

# Helper 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\_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

# LCD display cutout implementation

def add\_lcd\_cutout(enclosure\_obj, dimensions, position):

"""Add an LCD display cutout to the enclosure."""

try:

if not enclosure\_obj:

FreeCAD.Console.PrintError("Enclosure object not found\n")

return False

# Unpack dimensions

width, depth, height = dimensions

# Get the base shape

enclosure\_shape = enclosure\_obj.Shape

# Create LCD cutout - make it slightly larger than the LCD for proper fit

margin = 0.5 # 0.5mm margin on each side

cutout = create\_box(

width + margin\*2,

depth + margin\*2,

height + margin\*2,

Vector(

position[0] - margin,

position[1] - margin,

position[2] - margin

)

)

# Create LCD mounting frame

frame\_thickness = 2

frame\_depth = 1

frame = create\_box(

width + margin\*2 + frame\_thickness\*2,

frame\_depth,

height + margin\*2 + frame\_thickness\*2,

Vector(

position[0] - margin - frame\_thickness,

position[1] - frame\_depth,

position[2] - margin - frame\_thickness

)

)

# Create protective transparent cover (slightly recessed)

cover\_recess = 0.5

cover = create\_box(

width + margin\*2,

0.5, # Thin transparent cover

height + margin\*2,

Vector(

position[0] - margin,

position[1] - cover\_recess,

position[2] - margin

)

)

# Cut the enclosure with the LCD cutout

if cutout and enclosure\_shape:

enclosure\_shape = enclosure\_shape.cut(cutout)

# Add the frame by fusion

if frame and enclosure\_shape:

enclosure\_shape = enclosure\_shape.fuse(frame)

# Update the enclosure shape

if enclosure\_shape:

enclosure\_obj.Shape = enclosure\_shape

# Create transparent cover as a separate object

if cover:

cover\_obj = create\_object(cover, "LCDCover")

if cover\_obj:

# Set transparency for the cover

try:

cover\_obj.ViewObject.ShapeColor = (0.8, 0.8, 1.0) # Light blue tint

cover\_obj.ViewObject.Transparency = 80 # 80% transparent

except:

pass

return True

except Exception as e:

FreeCAD.Console.PrintError(f"Error adding LCD cutout: {e}\n")

return False

# Main implementation to add LCD cutout to an existing enclosure

def add\_lcd\_display\_to\_enclosure():

"""Add LCD display and cutout to the enclosure."""

try:

# Find the enclosure object

enclosure\_obj = None

for obj in doc.Objects:

if obj.Name == "IoTEnclosure":

enclosure\_obj = obj

break

if not enclosure\_obj:

FreeCAD.Console.PrintError("IoT enclosure not found\n")

return False

# IoT enclosure dimensions (should match the original enclosure)

iot\_enclosure\_width = 45

iot\_enclosure\_depth = 25

iot\_enclosure\_height = 25

iot\_enclosure\_wall = 2

# Position (in new standalone model, we can center it)

position\_x = 0

position\_y = 0

position\_z = 0

# LCD Display dimensions

lcd\_width = 35

lcd\_depth = 3

lcd\_height = 15

# LCD position - should be on the front face of the enclosure

lcd\_pos\_x = position\_x + (iot\_enclosure\_width - lcd\_width) / 2 # Center horizontally

lcd\_pos\_y = position\_y # At the front wall

lcd\_pos\_z = position\_z + (iot\_enclosure\_height - lcd\_height) / 2 # Center vertically

# Add the LCD cutout to the enclosure

result = add\_lcd\_cutout(

enclosure\_obj,

(lcd\_width, lcd\_depth, lcd\_height),

(lcd\_pos\_x, lcd\_pos\_y, lcd\_pos\_z)

)

# Create the LCD display itself (now visible through the cutout)

lcd = create\_box(

lcd\_width,

lcd\_depth,

lcd\_height,

Vector(

lcd\_pos\_x,

lcd\_pos\_y + iot\_enclosure\_wall - lcd\_depth, # Position just inside the front wall

lcd\_pos\_z

)

)

if lcd:

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

# Set LCD display color

try:

lcd\_obj.ViewObject.ShapeColor = (0.1, 0.1, 0.3) # Dark blue for LCD

except:

pass

# Recompute document

doc.recompute()

FreeCAD.Console.PrintMessage("LCD display and cutout added successfully!\n")

return True

except Exception as e:

FreeCAD.Console.PrintError(f"Error implementing LCD display: {e}\n")

return False

# Execute the function

add\_lcd\_display\_to\_enclosure()

# Display the model

try:

import FreeCADGui

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

FreeCADGui.SendMsgToActiveView("ViewFit")

FreeCADGui.activeDocument().activeView().viewAxometric()

except:

pass

print("LCD display cutout script completed!")