CASE STUDY CYBER PHYSICAL PRODUCTION SYSTEMS USING AM

SINGLE SCREW PUMP

Guide for Developing Single screw pump Components with 3D Printing using LUA Script in IceSl

GROUP 14

Gokul Kallingapuram Manoharan (22210892)

Athul Dev Nedumparambil Prasenan (22212087)

Jerry Jacob (22211985)

Research advisor:

Prof. Dr. -Ing. Stefan Scherbath

TECHNISCHE HOCHSCHULE DEGGENDORF

Table of Contents

List Of Figures

INTRODUCTION

This tutorial assists in the development of Single screw pump components for 3D printing using Lua script in IceSl. This guide leads users throughout the development of essential parts necessary to assemble a Single Screw Pump assembly with customizable parameters.

This guide is designed with the goal of allowing the user to comprehend various parameters and replicate the results.

The entire parametrical model is demonstrated within the Lua platform. Instructions for extracting necessary '.stl' files and G-codes for cross-platform applications and 3D printing of the components are also given in the documentation.

The demonstrated model includes modelling of mainly two components of Single Screw Pump, Rotor and Stator.

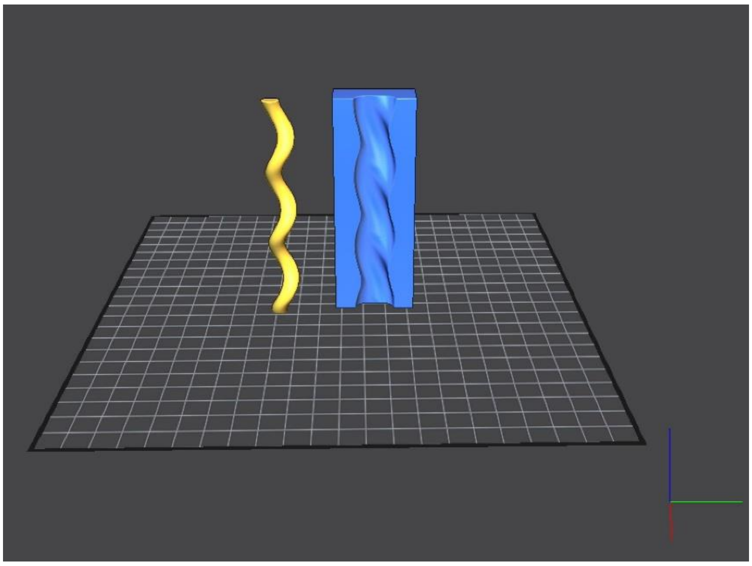


Figure 1: Rotor And Stator

The complete model with addition to rotor and stator includes inlet, outlet, tube flange, handle and stands.

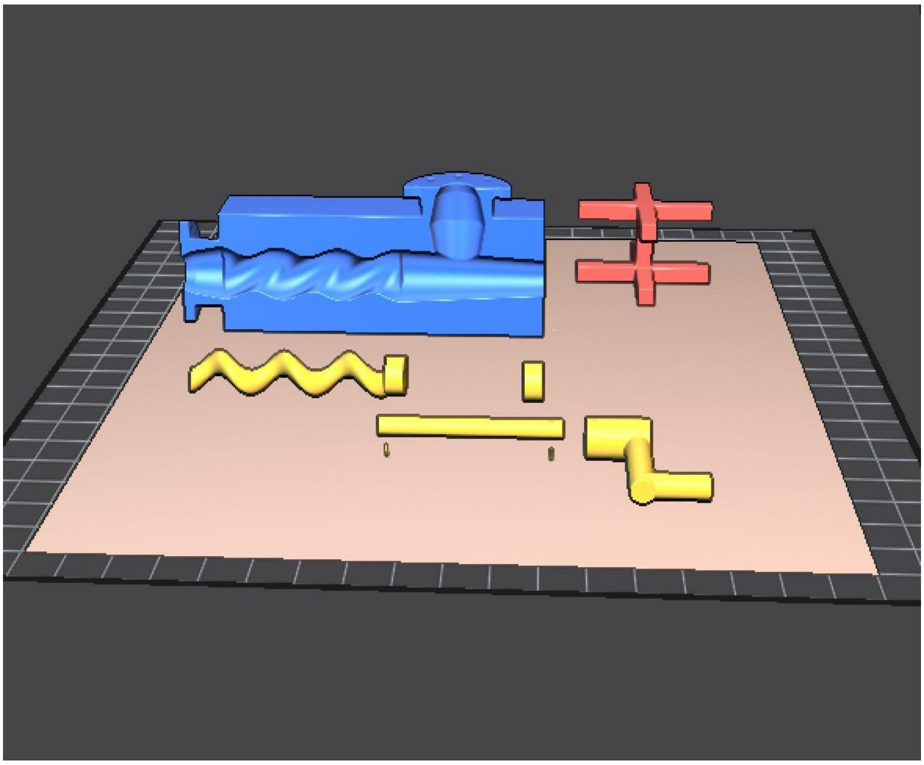


Figure 2: Cross-sectional view of pump

SCRIPT OVERVIEW

The script “Single_Screw_Pump.lua” allows user to generate the entire single screw pump model with user set parameters and 3D print them accordingly. The .lua file can be read and modified with editors supported by Icesl Forgeor Icesl Slicer.

This guide is divided into three parts:

1. Input parameters
2. Stator function
3. Rotor function
4. Extrude function
5. Flange function
6. Other components

INPUT PARAMETERS

Defining parameters is an important step. The tweak box in icesl takes input from the user and provides real time changes in the 3D model.

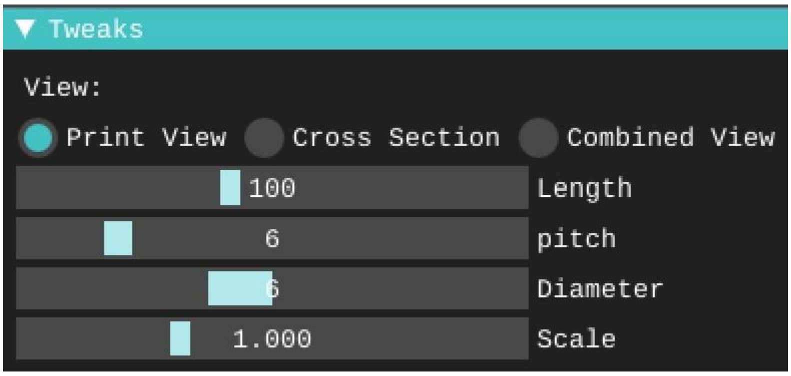


Figure 3: Tweak Box

The default values and the upper and lower limits of the tweak box can be adjusted in the user interface section of the script.

```
--------------------------------------------------User Interface--------------------------------------------------------------------

View_list = {
    {0, "Print View"},
    {1, "Cross Section"},
    {2, "Combined View"}
}

      --Creates a button to select an existing file. The return value is the file path

view = ui_radio("View:", View_list)

      --Creates a combo box to interactively choose a string value from a list.

len = ui_number("Length", 100, 30, 200)  --Length of the pump with default value of 100, upper limit of 200 and lower limit of 30
pit = ui_number("pitch", 6, 3, 20)       --Pitch of the pump with default value of 6, upper value of 10 and lower value of 3
diameter = ui_number("Diameter", 6, 3, 10) -- Diameter of the Rotor with default value of 6, upper limit of 10 and lower limit of 3
sc = ui_scalar("Scale", 1, 0.1, 3)          --Global scaling of the pump

      --Creates a sliding bar to interactively set an integer value between min and max.
```

Figure 4: User Interface

Function Stator_shape

The cross-sectional shape of the stator is two semicircles joined by tangential lines and stator is double helical in geometry. The function creates the shape by making two semicircles with an offset between them and joining the ends with a straight line. The shape is stored as a contour which will be extruded to obtain 3D shape. To adjust the distance between the semicircles, offset_length can be varied, by default it is set to radius of the semicircle.

```lua
----------------------------------------------Stator shape function--------------------------------------------------------------
function stator_shape(offset_length, d, n_points)


    --A function is defined which takes 3 arguments, diameter of semi circle and number of points by which the semi circle is plotted
    --The function creates two semicircles which are opposite to eachother and their end points are joined together

    local p1_XY = {}
    local p2_XY = {}

    --Two open arrays are declared to hold values
    local cos_phi, sin_phi
    for n = 1, n_points do
        local phi = math.pi * (n - 1) / n_points
        cos_phi = math.cos(phi)
        sin_phi = math.sin(phi)
        p1_XY[n] = translate(0, -len2, 0) * v(cos_phi * d, sin_phi * d, 5)
        p2_XY[n] = rotate(0, 0, 180) * translate(0, stator_offset, 0) * v(cos_phi * d, sin_phi * d)


      --A for loop is initialised in which n is iterated from 1 to n_points
    --phi is an angle that varies in the loop
    --p1_XY generates a semicircle
    --p2_XY generates another semicircle which is opposite to p1_XY
    end

    p1_XY[1] = p2_XY[#p2_XY]
    p1_XY[#p1_XY] = p2_XY[1]
    for n = 1, #p2_XY - 1 do
        table.insert(p1_XY, p2_XY[n])
    end

      --Two end points of the semicircles are joined to create a closed contour

    return p1_XY
      --The values are returned to p1_XY
end
```

Figure 5: Stator shape

Function Rotor_shape

The Rotor is helical in shape and the cross-sectional shape of the rotor in circular, The function creates a circle in 2D and the contour is stored in p1_XY

```lua
----------------------------------------------Rotor Function-------------------------------------------------------------
function rotor_shape(length, d, n_points)

      --The function rotor_shape creates circle which will be used to create the rotor
    local p1_XY = {}


    local cos_phi, sin_phi
    for n = 1, n_points do
        local phi = 2*math.pi * (n - 1) / n_points
        cos_phi = math.cos(phi)
        sin_phi = math.sin(phi)
        p1_XY[n] = translate(0, len2, 0) * v(cos_phi * d, sin_phi * d, 5)

    end
    -- Circle is plotted an the values are stored in p1_XY
    return p1_XY

    --The values are returned to p1_XY
end
```

Figure 6: Rotor shape

Function Extrude

The extrude function takes four arguments namely, contour, angle, dir_v and z_steps.

The double helical stator shape and the single helical rotor shapes are extruded in 3D with the help of extrude function. The stator_shape is called inside the extrude function as the contour and the angle direction and number of steps are given to obtain the double helical stator shape.

The helical shape of rotor is also obtained by passing the rotor_shape contour to the extrude function. The necessary parameters are given and the helical rotor is made.

```lua
-------------------------------------------------Extrude Function------------------------------------------------------------------------
function extrude(Contour, angle, dir_v, z_steps)

    --The function extrudes takes four arguments
   -- Contour, to extrude a Contour to a shape by turning the contour to angle in z_steps
   -- extrude a Contour in a dircetion given by the vector dir_v
   -- extrude a Contour with a scaling factor given by vector scale_v
   -- Contour: a table of vectors as a closed contour (start point and end point is the same)
   -- angle: roation angle of contour along z_steps in deg
   -- dir_v: vector(x,y,z) direction of extrusion
   -- z_steps: number of steps for the shape, mostly z_steps=2 if angle is equal zero
    local angle_rad = math.rad(angle)
    local Contours = {}

    -- n counter over contour points
    for n = 1, z_steps do
        local phi = angle_rad * (n - 1) / (z_steps - 1)
        local dir_vh = dir_v * (n - 1) / (z_steps - 1)
        local mod_contour = {}
        -- loop over contour points without end point because start is end point
          -- Calculate contur of vertex points by roating and scaling
        for i = 1, #Contour - 1 do
            local x = Contour[i].x
            local y = Contour[i].y
            local modified_vertex = v(
                dir_vh.x + (x * math.cos(phi) - y * math.sin(phi)),
                dir_vh.y + (x * math.sin(phi) + y * math.cos(phi)),
                dir_vh.z
            )
            mod_contour[i] = modified_vertex
        end
 -- end loop over points of contour
        mod_contour[#mod_contour + 1] = mod_contour[1]
        Contours[n] = mod_contour

-- Calc. the modified contour for a z_level
    end

    return sections_extrude(Contours)
-- return shape
end
```

Figure 7: Function Extrude

Function Flange

Two flanges are used at the inlet and outlet of the pump, which can be used for mounting. The flange can be modified with this function as per the requirement of the user. Conical path ways are used in the model.

```lua
-------------------------------------------------Flange Function------------------------------------------------------------------------
function flange(dia_hole,dia_big,hole_num,hole_circle)
    --This function creates the tube flange which is used at the inlet and outlet
  local base = translate(0,0,0)*cylinder (dia_hole*1.8,8)
    -- The base of the flange used is a cylinder
  local plate = translate(0,0,8)*difference(cylinder(dia_big,3), cone (dia_hole*1.2,dia_hole/1.5,5))

    --The plate that will be used to mount the screws
  local cut = cone (dia_hole*1.5,dia_hole*1.2,8)


  local flange_full= union(base, plate)
  local flange_full = difference(flange_full, cut)


  local circle_radius = hole_circle -- Radius of the screw circle
  local num_holes = hole_num -- Number of screw holes to emit

  local angleStep = (2 * math.pi) / num_holes

  local cylinders = {} -- Table to store the smaller cylinders

  for i = 1, num_holes do
    local angle = (i - 1) * angleStep
    local x = circle_radius * math.cos(angle)
    local y = circle_radius * math.sin(angle)
    local z = 0
    local cylinder1 = translate(x, y, 8) * cylinder(1, 5)
  table.insert(cylinders, cylinder1)
  end

  --A loop iterates points of a circle in which the holes will be placed

  for i, cylinder1 in ipairs(cylinders) do
    flange_full = difference(flange_full, cylinder1) -- Subtract each smaller cylinder from the result
  end
  return flange_full
end
```

Figure 8: Function Flange

Other Components

Connecting rod housing and inlet port

The connecting rod is housed inside a conical cavity in which the liquid from inlet gets in and is then pumped.

```
inlet_way = translate(0,-1,24)*rotate(-90,0,0)*cone(diameter,diameter*1.8,25)

    --the inlet hole for the inlet


inlet_base= cube(diameter+30,diameter+30,40)
inlet_cut= rotate(0,0,0)*cone(diameter,diameter*1.5,40)
inlet_housing = difference(inlet_base,inlet_cut)
inlet_housing = difference(inlet_housing,inlet_way)
stator = rotate(90,0,0)*union(stator,inlet_housing)

    --The part where the inlet housing is made
```

Figure 9: inlet housing

Rotor bearing

The rotor is fixed with a bearing to which the connecting shaft is joined.

```
bearing1= difference(translate(0,-35,-1)*rotate(90,0,0)*cylinder(diameter,5),translate(0,-35,-1)*rotate(90,0,0)*
cylinder(diameter/2,3.5))
  -- bearing to connect rotor and connecting shaft
rotor= union(rotor,bearing1)
```

Figure 10:Bearing for rotor

Connecting Rod bearing

A bearing at the end of connecting rod is placed to support the connecting rod.

```
function bearing2()
  -- Bearing to hold the connecting shaft
  local b1=translate(0,0,0)*rotate(0,0,0)*cylinder(diameter*0.99,5)

  local b2=translate(0,2,0)*rotate(0,0,0)*cylinder(diameter/2,5)
  local b3=translate(0,-2,0)*rotate(0,0,0)*cylinder(diameter/2,5)

  local b4= translate(0,0,0)*union (b2,b3)
  local b4= convex_hull(b4)

  local bearing2 = rotate(90,0,0)*difference(b1,b4)
  return bearing2
end
```

Figure 11:Bearing for connecting rod

Handle

A handle is designed to simulate the working of pump, that acts as the driving mechanism.

```
function handle()
  -- creates a handle which acts as driving mechanism
  local cy1=difference(cylinder(diameter,15),cylinder(diameter/2,10))
  local cy2= translate(0,0,12)*rotate(90,0,0)*cylinder(3,25)

  local cy3 = translate(0,-22,12)*rotate(0,0,0)*cylinder(3,15)

  local tt = union{cy1,cy2,cy3}
  local handle= difference(tt, translate(-7,0,2)*rotate(0,90,0)*cylinder(.5,15))
  local handle =rotate(0,90,90)*handle
  return handle
end
```

Figure 12: Handle

Keys

Two keys in the shape of cylinder are used to keep the connecting rod in place.

```
key1 = translate(0,-37,-8)*cylinder(0.3,3)
key2 = translate(0,2,-5)*cylinder(0.3,3)

  --key for the connecting shaft
```

Figure 13: keys for connecting rod

Stand

Stand is made using cubes stacked in a particular way. The stand is provided to successfully display the pump.

```
function stand(width,height)

    -- creates a stand to support the pump
  local mid= cube(width,width,height)
  local low1 = translate(-20,0,0)*rotate(0,90,0)*cube(width,width,40)
  local low2 = translate(0,20,0)*rotate(90,0,0)*cube(width,width,40)
  local hig1 = translate(-20,0,height)*rotate(0,90,0)*cube(width,width,40)
  local hig2 = translate(0,20,height)*rotate(90,0,0)*cube(width,width,40)
  local stand = union{mid,low1,low2,hig1,hig2}
return stand
end
```

Figure 14: Stand

SLICING AND G-CODE

The 3D model can be exported as ".stl" file or G-code, for 3D printing according to the user preference. The step by step instruction below would be helpful for it.

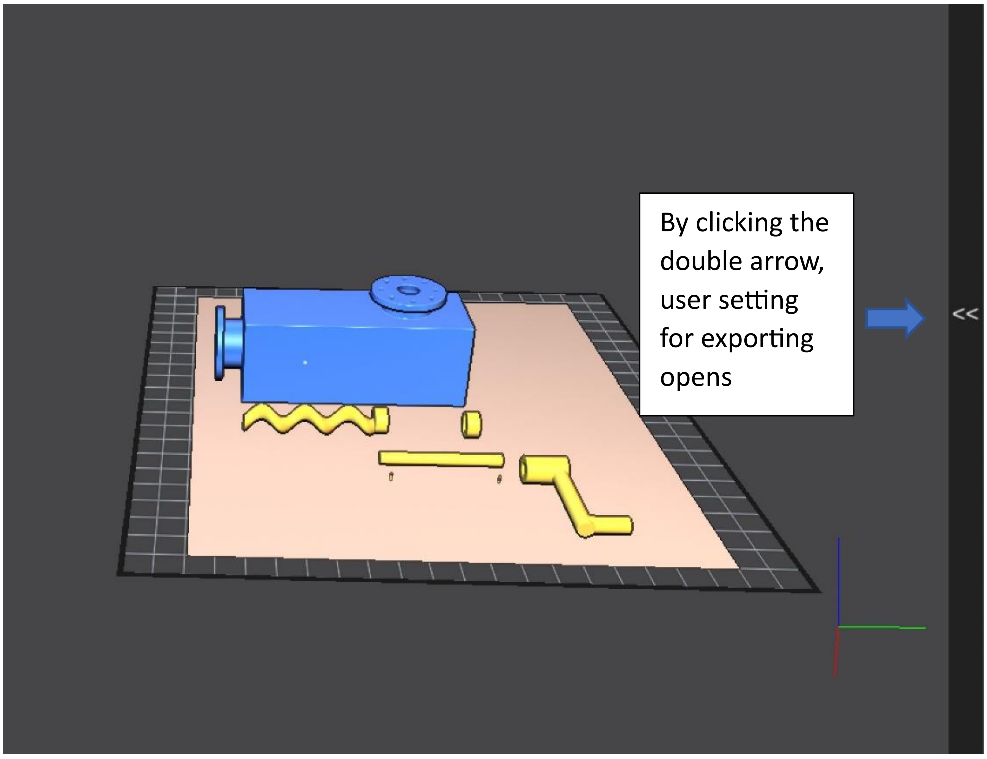


Figure 15: Slicing step 1

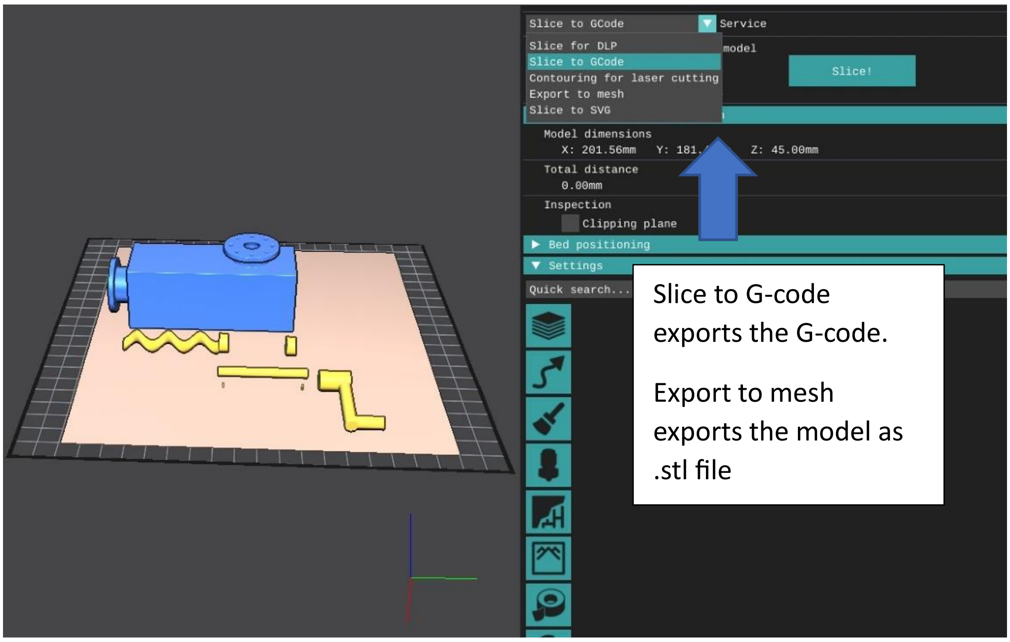


Figure 16: Slicing Step 2

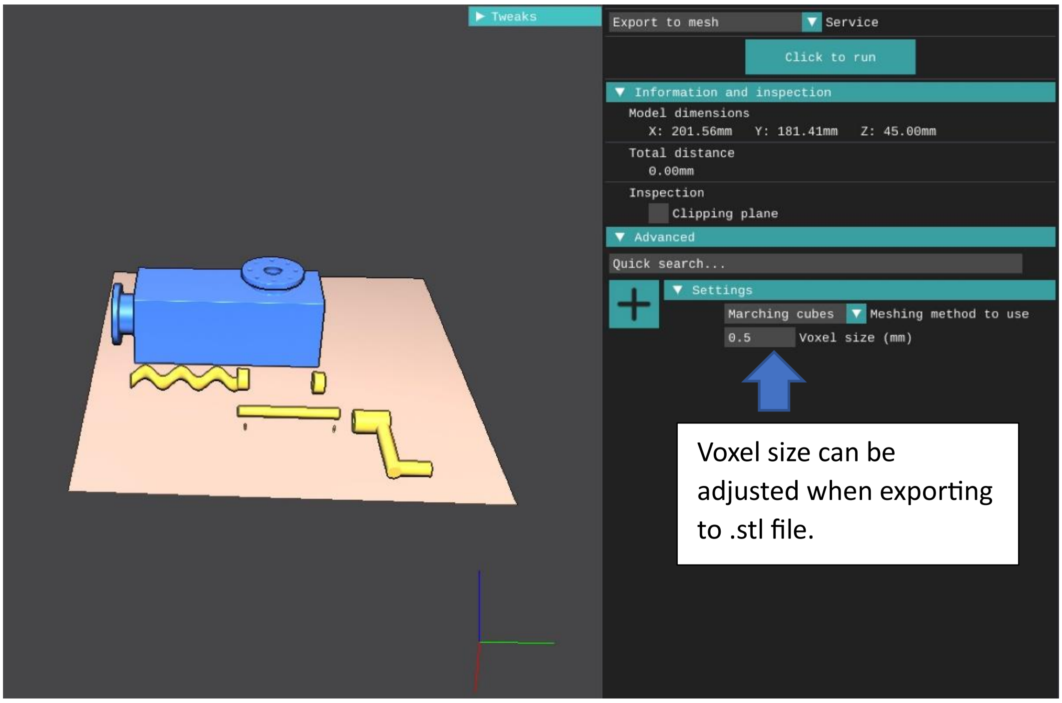


Figure 17: Adjusting voxel size

ASSEMBLY OF THE PARTS

The assembled view can be seen when the "combined view" button in the tweak box in engaged. A hint for assembly of the printed parts can be seen in the following figures below

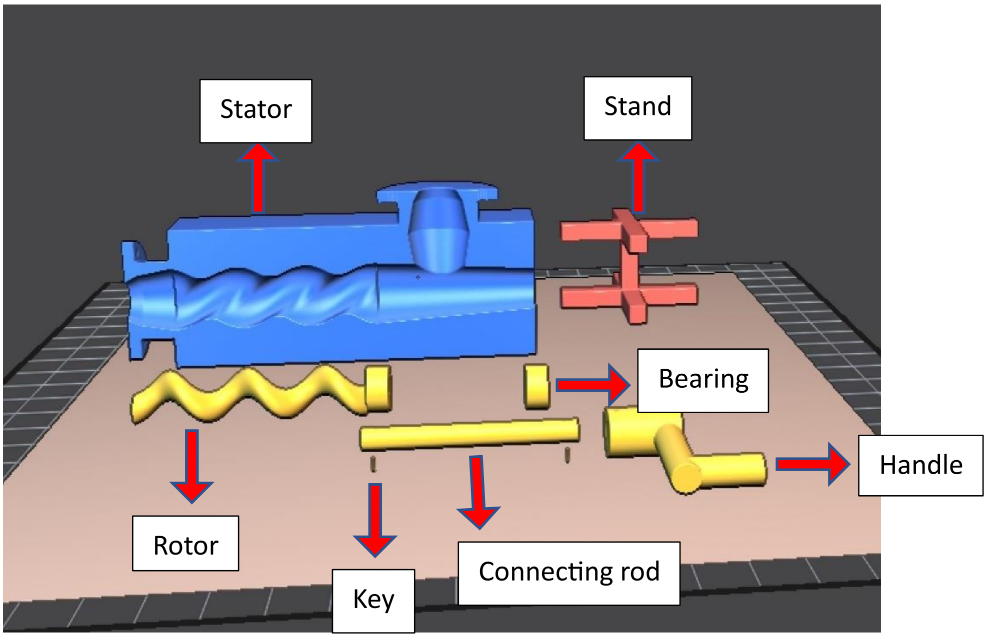


Figure 18: Components

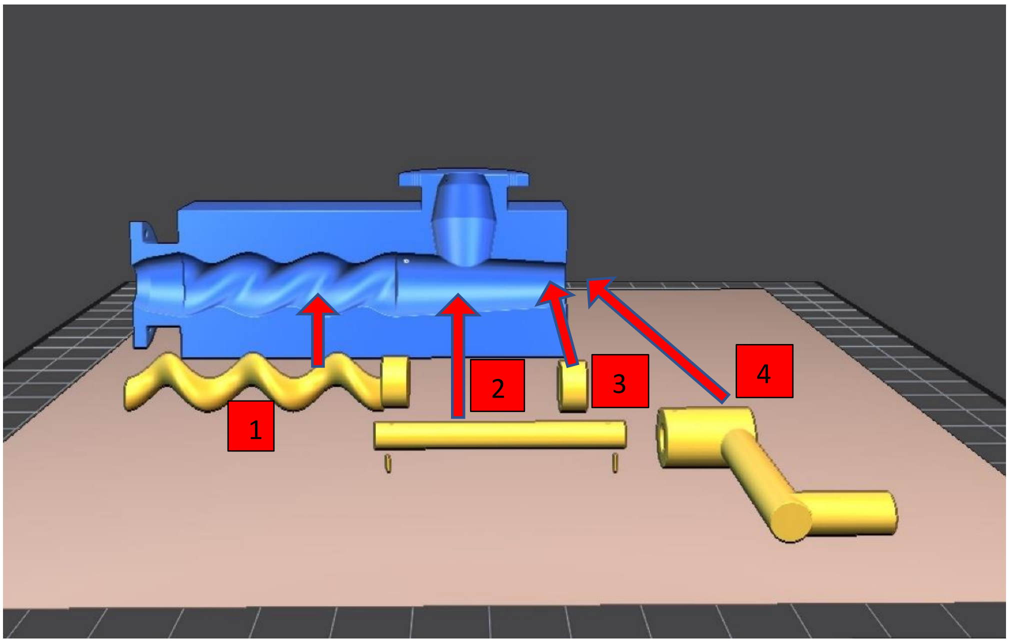


Figure 19: Assembly step 1

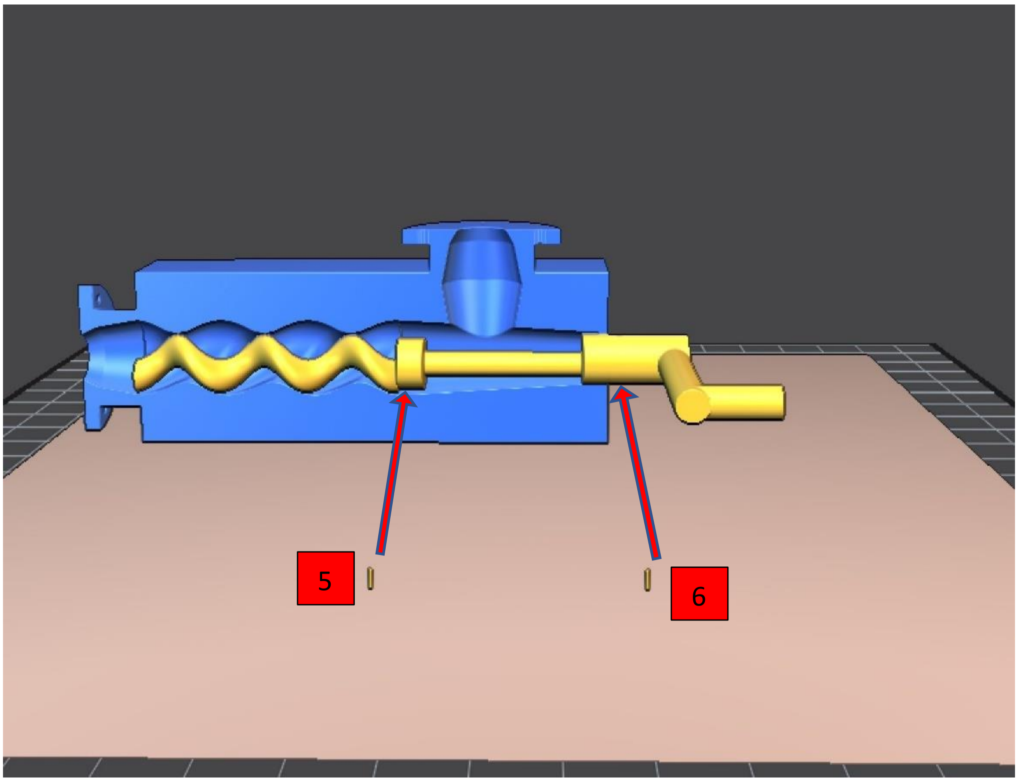


Figure 20: Assembly step 2

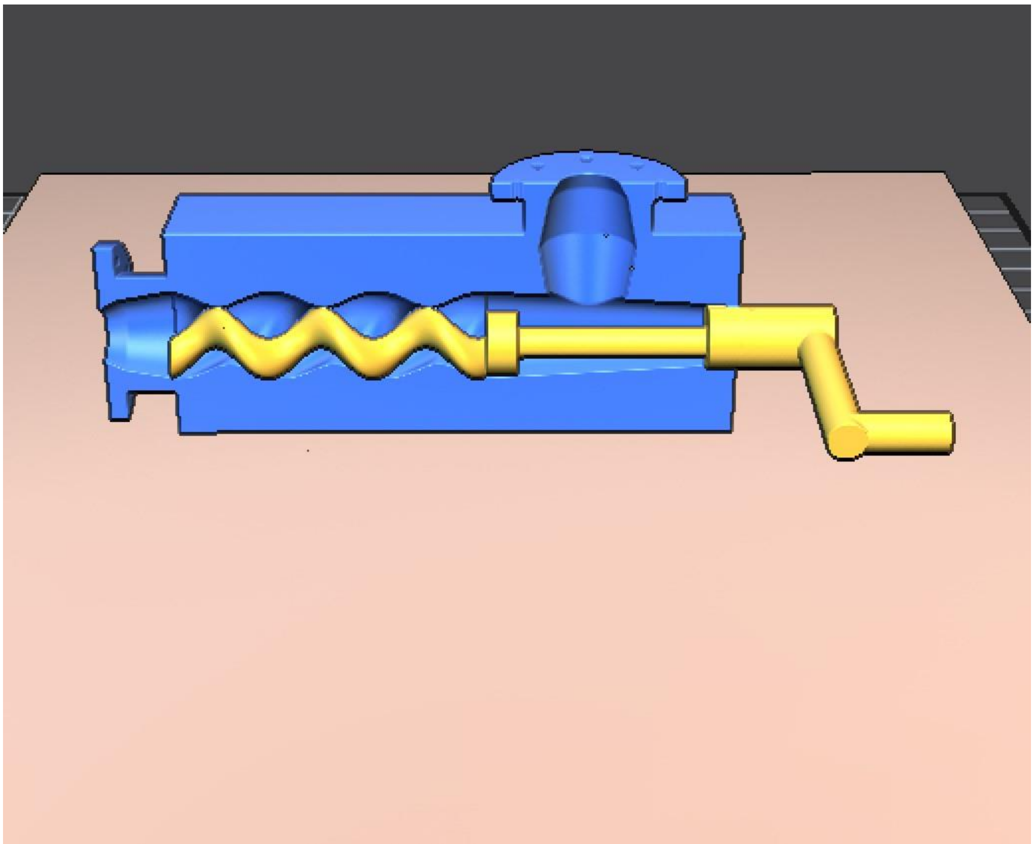


Figure 21: Full assembly

The red arrows shows where each component goes, rotor is placed inside the stator, The connecting rod support bearing is placed at the end of the stator, the connecting rod is then fixed in the rotor bearing. The handle for turning the rotor is fixed with the hole aligned. Two keys are then used to lock the connecting rod and handle.

Finally, the pump is placed upon a stand.

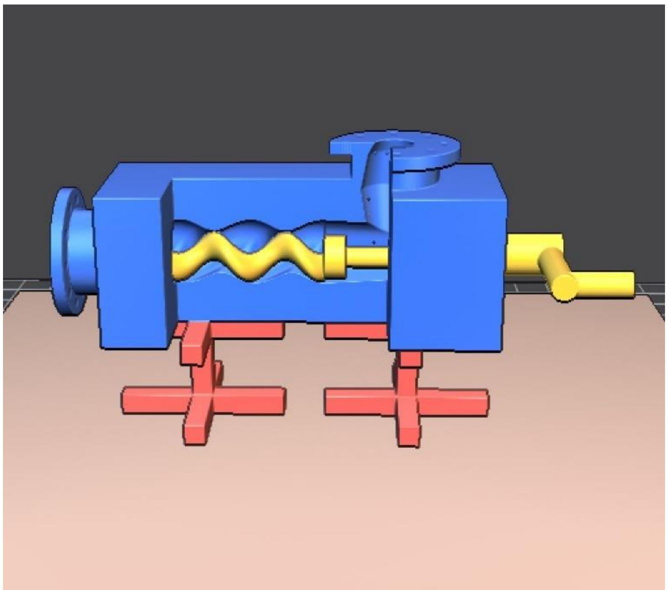
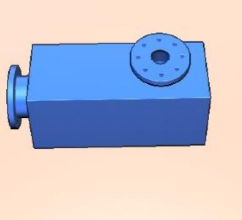
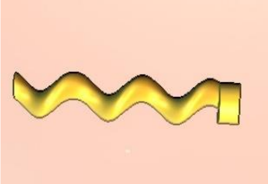
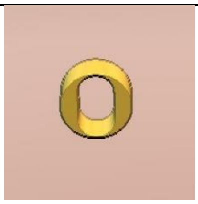
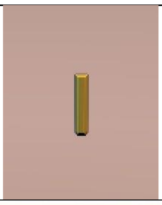
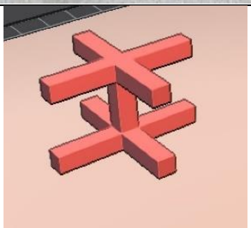


Figure 22: Presentation view

LIST OF PARTS

Name of the part	Quantity	Picture
1. Stator with Inlet and Outlet flange.	1	
2. Rotor	1	
3. Connecting rod	1	
4. Bearing	1	
5. Handle	1	
6. Key	2	
7. Wooden plate A3 size	1	

8. Spax Screws	8	
9. Stand	2	
10. Glue	1	